İçindekiler

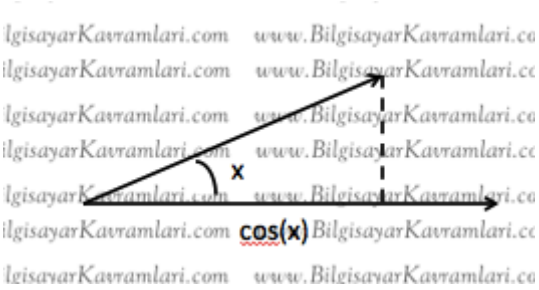
DOSYA ORGANİZASYONU

SORU-1: SimHash (Benzerlik Özeti) hakkında bilgi veriniz.

Bilgisayar bilimlerinde, özellikle metin işlemenin yoğun olduğu, arama motoru gibi uygulamalarda dosyaların veya web sitelerinin birbirine olan benzerliğini bulmak için kullanılan bir algoritmadır.

Algoritmaya alternatif olarak klasik hash fonksiyonları kullanılabilir. Yani, örneğin iki sayfasnın ayrı ayrı hash değerleri alınıp bu değerleri karşılaştırmak mümkündür. Ancak simhash algoritması, bu yönteme göre daha fazla hız ve performans sunar.

Sim hash algoritması, iki dosyayı birer vektör olarak görür ve bu vektorler (yöney, vector) arasındaki cosinüs (cosine) bağlantısını bulmaya çalışır.



Yukarıdaki şekilde temsil edildiği üzere iki dokümanın ayrı ayrı birere vektör olması durumunda, aralarında cos (x) olarak gösterilen bir açı ile bağlantı bulunması mümkündür.

Algoritma, öncelikle işlediği metindeki kelimelerin ağırlıklarını (weight) çıkarmakta ve buna göre kelimeleri sıralamaktadır.

Sıralanan her kelimeye, b uzunluğunda, yegane (unique) değer döndüren bir fonksiyon kullanılır. Örneğin her kelime için farklı bir hash sonucu döndüren fonksiyon kullanılır.

b boyutundaki bir vektörün ağırlık değeri hesaplanırken, her kelimedeki 1 değeri için +1 ve 0 değeri için -1 değeri ağırlığa eklenir.

Son olarak üretilen ağırlık vektöründeki + değerler 1, 0 ve – değerler ise 0 olarak çevirilir.

Örnek

Yukarıdaki algoritmanın çalışmasını bir örnek üzerinden anlatalım. Algoritmanın üzerinde çalışacağı metin aşağıdaki şekilde verilmiş olsun:

www bilgisayar kavramları com bilgisayar kavramlarının anlatıldığı bir bilgisayar sitesidir ve com uzantılıdır

Yukarıdaki bu metni, algoritmanın anlatılan adımlarına göre işleyelim:

İlk adımımız, algoritmadaki kelimelerini ağırlıklarının çıkarılmasıdır. Bu adımı çeşitli şekillerde yapmak mümkündür ancak biz örneğimizde kolay olması açısından kelime frekanslarını (tekrar sayısı, frequency) kullanacağız. Buna göre metindeki kelimelerin tekrar sayılarına göre sıralanmış hali aşağıda verilmiştir:

bilgisayar 3 com 2 kavramları 1 kavramlarının 1 anlatıldığı 1 bir 1 www 1 sitesidir 1 ve 1 uzantılıdır 1

Yukarıda geçen her kelime için bir parmak izi (fingerprint) değeri üretiyoruz. Bu değerin özelliği, kelimeler arasında yegane (unique) bir değer bulmaktır. Bu değer, herhangi bir hash fonksiyonu üzerinden de üretilebilir. Biz örneğimizde kolalık olması açısından her kelime için rast gele bir değer kendimiz atayacağız. Ancak gerçek bir uygulamada rast gele değerlerin kullanılması mümkün değildir. Bunun sebebi, aynı kelimenin tekrar gelmesi halinde yine aynı değerin üretilmesi zorunluluğudur. Bu yazıdaki amaç algoritmayı anlatmak olduğu için birer hash sonucu olarak rast gele değerler kullanılacaktır.

bilgisayar 10101010 com 11000000 kavramları 01010101 kavramlarının 10100101 anlatıldığı 11101110 bir 01011111 www 11110001 sitesidir 10101110 ve 00001111 uzantılıdır 00100010

3. adımda, yukarıdaki değerleri topluyoruz. Toplama işlemi sırasında 1 değerleri için +1 ve 0 değerleri için -1 alıyoruz.

10101010
11000000
01010101
10100101
11101110
01011111
11110001
10101110
00001111
00100010
———
2 0 2 -4 0 2 2 0

Son olarak, yukarıdaki değerleri ikilik tabana çeviriyoruz: 10100110 bu değer bizi simhash sonucumuz olarak bulunuyor.

Örneğin yeni bir dosyayı daha işlemek istediğimizde, bu dosyadaki kelime yoğunluğuna göre yukarıda bulduğumuz simhash değerine yakın bir değer çıkmasını bekleriz.

Diyelim ki yeni bir dosyada da sadece “bilgisayar kavramları com” yazıyor olsun. Bu yazının sim hash değerini bularak karşılaştırmaya çalışalım:

bilgisayar 10101010 com 11000000 kavramları 01010101

10101010
11000000
01010101
———
1 1 -1 -1 -1 1 1 1

Değerin ikilik tabana çevrilmiş hali : 11000111

Orjinal dokümandan çıkardığımız simhash değeri ile farklı olan bit sayısı 3′tür. Bunun anlamı yukarıdaki bilgisayar kavramları com yazısının orjinal yazıya 3 mesafesinde yakın olduğudur.

SORU-2: Doğrusal Karım (Linear Hashing) hakkında bilgi veriniz.

Bu yazının amacı, doğrusal karım ve doğrusal karım tablosu (linear hash table) konularını anlatmaktır. Bilgisayar bilimlerinde veri depolamak veya veriye hızlı ulaşmak için kullanılan yöntemlerdir.

Doğrusal karım yönteminde temel olarak özetleme fonksiyonları kullanılır (karım fonksiyonu, hash function). Bu fonksiyonlar sıralıdır ve sayısı, özetleme fonksiyonunun seviyesini belirtir. (h1, h2, h3, … şeklinde istenilen seviyede istenildiği kadar fonksiyon olabilir).

Doğrusal karım'ın en önemli özelliği yük çarpanını (load factor) sabit tutmasıdır. Yük faktörü aşağıdaki formül ile hesaplanabilir:

LF = R / (B * c)

Buradaki R değeri bir kayıtta tutulan anahtar sayısı (Record),

B değeri, kovaların (Bucket) sayısı

c değeri ise bir bloktaki kayıt sayısıdır.

Doğrusal karım için bir anahtar değerine (key value) erişim aşağıdaki algoritma ile olur.

- Verilen seviye için özetleme fonksiyonuna sokulup özetleme değeri hesaplanır h = H(key) , key anahtar değeri, H() fonksiyonu özet fonksiyonu ve h değeri ise özetleme fonksiyonu sonucudur.
- Sınır değeri hesaplanır (Boundary value, kısaca bv) bv = c * LF , buradaki c değeri blok başınsa tutulan kayıt değeridir. LF ise yük çarpanı (load factor) değeridir.
- Sonucun son k hanesi için bir yerleştirme işlemi yapılır. (örneğin k değeri 3 ise, sonuçta bulunan değerin son 3 hanesine bakılarak yerleştirme işlemi yapılır)
- Genişleme gerekmesi durumunda, ilgili kova (bucket, buket) k+1 hane kullanarak bölünür.

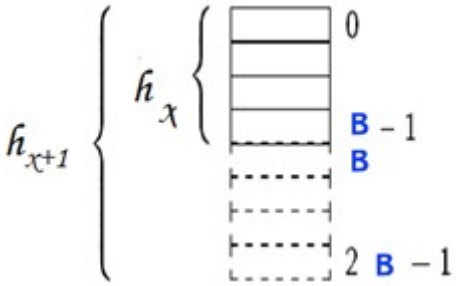
Yukarıdaki algoritmadan anlaşılacağı üzere, k değeri o andaki saklanan bilgiler için tutulan değerdir. Karım'a ekleme yapıldıkça k değeri artar (her ekleme ile birlikte artması gerekmez ama belirli değerlere ulaşıldığında artar).

Ayrıca doğrusal karım ile ilgili bilinmesi gereken bir nokta da özetleme fonksiyonlarının özelliğidir.

- Özetleme fonksiyonları arasında kesişim için özetleme fonksiyonuna sokulup özetleme değeri hesaplanır h = H(key) , key anahtar değeri, H() fonksiyonu özet fonksiyonu ve h değeri ise özetleme fonksiyonu sonucudur.
- Sınır değeri hesaplanır (Boundary value, kısaca bv) bv = c * LF , buradaki c değeri blok başınsa tutulan kayıt değeridir. LF ise yük çarpanı (load factor) değeridir.

- Sonucun son k hanesi için bir yerleştirme işlemi yapılır. (örneğin k değeri 3 ise, sonuçta bulunan değerin son 3 hanesine bakılarak yerleştirme işlemi yapılır)
- Genişleme gerekmesi durumunda, ilgili kova (bucket, buket) k+1 hane kullanarak bölünür.

olmamalıdır. Örneğin h_x fonksiyonu x seviyesindeki fonksiyon ise, h_{x+1} seviyesindeki fonksiyonun h_x fonksiyonunun etki alanının iki misline sahip olması gerekir.



Şekilde görüldüğü gibi, şayet h_x fonksiyonu, B-1 elemana kadar kodlayabiliyorsa, h_{x+1} fonksiyonunun 2B-1′e kadar kodlayabilmesi beklenir.

Yukarıdaki bu özelliği sağlayan en temel özetleme fonksiyonu aşağıdaki şekilde yazılabilir:

hx(key) = H(k) mod (2x B)

Bu fonksiyonda, hx ilgili seviyedeki özetleme fonksiyonu (karım fonksiyonu, hash function) olmak üzere, H(k) ise sistemimizde kullandığımız bir özetleme fonksiyonu olmak şartıyla (ve ayrıca sistemin ihtiyacı olan en büyük B değerinden daha büyük bir sonuç üretebilecek entropiye (Entropy) sahip olmalıdır). Sistemimiz, 2x değerini B ile çarparak kalan işlemine tabi tutmaktadır (modulo). Sonuçta çıkan değerler ilgili seviyeden B değerinin içerisine sığdırılmaktadır. Diğer bir deyişle yukarıdaki fonksiyonumuz B, 2B, 4B, 8B … şeklinde bir öncekinin iki misli olan kodlama aralığı üretmektedir.

Yukarıda anlatılan konuyu bir örnek üzerinden anlamaya çalışalım.

Örnek

Öncelikle özetleme fonksiyonlarımızı aşağıdaki şekilde tanımlayalım.

$h_0(k) = k \bmod 2^0$

$h_1(k) = k \bmod 2^1$

$h_2(k) = k \bmod 2^2$

$h_3(k) = k \bmod 2^3$

…

Örneği kolay tutmak için sadece ilgili değerin gerekli olan özetleme fonksiyonu seviyesinden 2 üzeri şeklinde modulo'su hesaplanmaktadır. Örneğin 0. seviyedeki bir özetleme işlemi her zaman 0 üretmekte , 1. Seviyeden bir özetleme fonksiyonu ise 0 veya 1 üretmektedir…

Doğrusal karım içerisine eklemek istediğimiz sayılar aşağıdaki şekilde verilmiş olsun :

45, 33, 78, 22, 21, 76, 23, 65

Bu sayıları, yukarıda tanımı yapılan fonksiyonlar ile doğrusal karım tablomuza eklemeye çalışalım.

Ekleme işlemi sırasında bize eşlik edecek önemli bir değişkenimiz bulunuyor. n = 0 olarak başlıyoruz. Buradaki n, bölmek için sıranın kaçıncı bukette (kova, bucket) olduğudur.

İlk gelen sayımız 45 ve biz bu sayının 0. seviyedeki özetleme fonksiyonu sonucunu alıyoruz ve doğal olarak 0 çıkıyor:

$h_0(45) = 45 \bmod 2^0 = 0$

Sıra	Değer
0	45

Yukarıda görüldüğü şekilde gelen yeni değeri tablonun 0. sırasına ekliyoruz. Sıradaki sayımız 33. Yukarıdaki doğrusal karım tablomuzda tek eleman koymak için yer bulunuyor ve bizim ise iki elemanımız bulunuyor. Bu durumda k değerini arttırıyoruz ve 2 eleman sığabilecek şekilde tablomuzu genişletiyoruz.

$h_0(33) = 33 \bmod 2^0 = 0$

$h_1(33) = 33 \bmod 2^1 = 1$

Buket No	Sıra	Değer
0	0	
	1	
1	2	45
	3	33

Yukarıdaki şekilde görüldüğü üzere 2 ayrı kova (bucket) oluşturulmuş ve anahtarların, bu kovalardan hangisine yerleştirileceğini belirlemek için h_1 fonksiyonu kullanılmıştır. Elimizdeki değerlerin ikisi de (45 ve 33) tek sayı olduğu için ikisi birden ikinci bukete yerleştirilmiştir.

Şimdilik herhangi bir çakışma olmadan sayılarımızı ekliyoruz. Sıradaki sayımız 78.

$h_0(78) = 78 \bmod 2^0 = 0$

$h_1(78) = 78 \bmod 2^1 = 0$

Buket No	Sıra	Değer
0	0	78
	1	
1	2	45
	3	33

Sıradaki sayımız 22:

$h_0(22) = 22 \bmod 2^0 = 0$

$h_1(22) = 22 \bmod 2^1 = 0$

Buket No	Sıra	Değer
0	0	78
	1	22
1	2	45
	3	33

Sıradaki sayımız 21

$h_0(21) = 21 \bmod 2^0 = 0$

$h_1(21) = 21 \bmod 2^1 = 1$

Ne yazık ki, burada bir çakışma oluyor. Bunun sebebi şu anda sayıyı yerleştirmek istediğimiz buketin dolu oluşudur. Hemen k değerini arttırarak yeni bir buket oluşturuyoruz ve özetleme fonksiyonunu bir seviye ilerletiyoruz.

Şu anda n değerimiz 0 (örneğin başında atamıştık). Bunun anlamı bölme işleminin uygulanacağı buketin 0. buket olduğu. O halde yeni yer açmak için bu buketi parçalıyoruz ve aşağıdaki şekilde bir doğrusal karım tablosu elde ediyoruz. Ayrıca n değerini artık 1 olarak arttırıyoruz. Yani sıradaki parçalama işlemi 1. bukete yapılacak.

$h_0(21) = 21 \bmod 2^0 = 0$

$h_1(21) = 21 \bmod 2^1 = 1$

$h_2(21) = 21 \bmod 2^2 = 1$

Buket No	Sıra	Değer	Genişleme
00	0	78	
	1	22	
1	2	45	
	3	33	21
10	4		
	5		

Yukarıda bir genişleme işlemi yapılmıştır. Genişletilen buket ise 00 numaralı bukettir. Bu durumda 00 numaralı bukete bulunan değerlerin durumu yeniden kontrol edilip bölünmeden sonra yerlerinde mi kalacakları yoksa yeni bukete mi gidecekleri sorgulanmalı:

$h_0(78) = 78 \bmod 2^0 = 0$

$h_1(78) = 78 \bmod 2^1 = 0$

$h_2(78) = 78 \bmod 2^2 = 2$

$h_0(22) = 22 \bmod 2^0 = 0$

$h_1(22) = 22 \bmod 2^1 = 0$

$h_2(22) = 22 \bmod 2^2 = 2$

Yukarıdaki işlemlerden sonra tablomuz aşağıdaki hale gelir:

Buket No	Sıra	Değer	Genişleme
00	0		
	1		
1	2	45	
	3	33	21
10	4	78	
	5	22	

Sıradaki sayımız 76. Bu sayı için de özetleme fonksiyonlarımızı kullanıyoruz:

$h_0(76) = 76 \bmod 2^0 = 0$

$h_1(76) = 76 \bmod 2^1 = 0$

$h_2(76) = 76 \bmod 2^2 = 0$

Buket No	Sıra	Değer	Genişleme
00	0	76	
	1		
1	2	45	
	3	33	21
10	4	78	
	5	22	

Sıradaki sayımız 23. Bu sayı için özetleme fonksiyonları kullanıldığında yeni bir genişleme ihtiyacı doğacaktır:

$h_0(23) = 23 \bmod 2^0 = 0$

$h_1(23) = 23 \bmod 2^1 = 1$

$h_2(23) = 23 \bmod 2^2 = 3$

Genişleme işleminin uygulanacağı buket ise n ile gösterilmekte olan 1 numaralı bukettir. Buket genişletildikten sonra, tablomuz aşağıdaki hale dönüşür:

Buket No	Sıra	Değer	Genişleme
00	0	76	
	1		
01	2	45	
	3	33	21
10	4	78	
	5	22	
11	6	23	
	7		

Yukarıdaki tabloda işlem henüz tamamlanmamıştır. Bunun sebebi genişleme yapılan buketin içerisinde bulunan değerlerin kontrol edilerek, mevcut yerlerinde mi kalacakları yoksa yeni bukete mi gideceklerine bakılmamış olmasıdır.

$h_0(45) = 45 \bmod 2^0 = 0$

$h_1(45) = 45 \bmod 2^1 = 1$

$h_2(45) = 45 \bmod 2^2 = 1$

$h_0(33) = 33 \bmod 2^0 = 0$

$h_1(33) = 33 \bmod 2^1 = 1$

$h_2(33) = 33 \bmod 2^2 = 1$

$h_0(21) = 21 \bmod 2^0 = 0$

$h_1(21) = 21 \bmod 2^1 = 1$

$h_2(21) = 21 \bmod 2^2 = 1$

Yukarıdaki denklemlerden görüldüğü üzere sayılarımız mevcut yerlerinde kalacaklardır. Unutmamamız gereken değişim ise n değerini arttırarak artık 2 yapmaktır.

Sıradaki sayımız 65:

$h_0(65) = 65 \bmod 2^0 = 0$

$h_1(65) = 65 \bmod 2^1 = 1$

$h_2(65) = 65 \bmod 2^2 = 1$

Buket No	Sıra	Değer	Genişleme
00	0	76	
	1		
01	2	45	65
	3	33	21
10	4	78	
	5	22	
11	6	23	
	7		

Yukarıdaki tabloda görüldüğü üzere bütün değerler tabloya yerleştirilmiştir. Tablonun performansı ile ilgili olarak en kötü durumu düşünelim ve bundan sonraki gelecek sayıların sürekli olarak 01 buketine geldiğini hayal edelim. Bu durumda tablo bölünmeye devam edecek ve en kötü ihtimalle 3 dönüşten sonra 01 buketini parçalayacaktır. Dolayısıyla buketlerde taşma olması halinde (overflow) belirli bir sürede bu bukete dönülmekte ve bölme işlemi yapılarak tablodaki yük dengesi sağlanmaktadır.

Yukarıdaki bu yazıyı yazmama sebep olan, Hande hanımın sorusunu bu bilgilerden sonra cevaplayayım. Öncelikle soruyu alıntılayalım:

Sınır değerinin bv = $(10110)_2$, k değerinin 5 olduğu bir doğrusal kırım tablosunda (linear hash table), birincil bölgede kaç tane kova (bucket) vardır?
A) 5 B) 22 C) 32 D) 54 E) 76

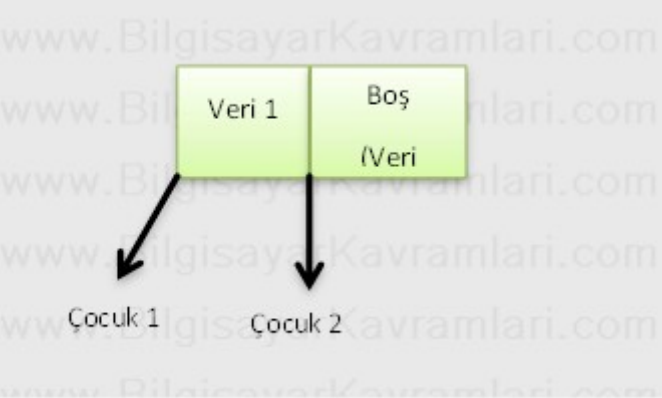
Sorudaki sınır değeri (boundary value , bv) 10110 olarak verilmiş. Bu değerin 10′luk tabandaki karşılığı 22′dir. k değerinin 5 olmasından anlaşılacağı üzere, özetleme fonksiyonlarımız o anda son 5 haneye bakmaktadır. Son 5 haneye bakarak kodlayabileceğimiz kova sayısı $2^5 = 32$ olarak bulunur. Ayrıca sınır değeri olarak verilen 22 kovaya da erişebildiğimizi düşünürsek, 22 + 32 = 54 olarak anlık erişilebilen kova sayısı (bucket) bulunmuş olur.

SORU-3: 2-3-4 Ağaçları (2 3 4 trees) hakkında bilgi veriniz.

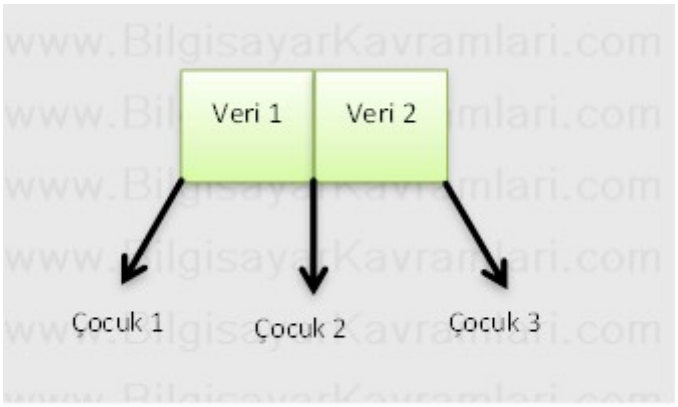
2-3-4 ağacı, B-ağaçlarının (B-Trees) özel bir halidir. Bu ağacın özelliği, düğüm boyutunun (node size) 3 ile sınırlı olmasıdır. Ağaç ayrıca sürekli olarak dengeli bir ağaç garantisi verir (balanced tree). 2-3-4 ağaçları, kırmızı siyah ağaçlarının (red-black trees) , eş şekillisi (isomorphic) olarak da düşünülebilir.

2-3-4 ağacının ismi, ağaçtaki düğümlerin değişik durumlarda değişik numaralar almasından kaynaklanmaktadır. Yani bir düğüm 2,3 veya 4 durumunda olabilir. Bu durumlara göre farklı uygulamalar söz konusudur.

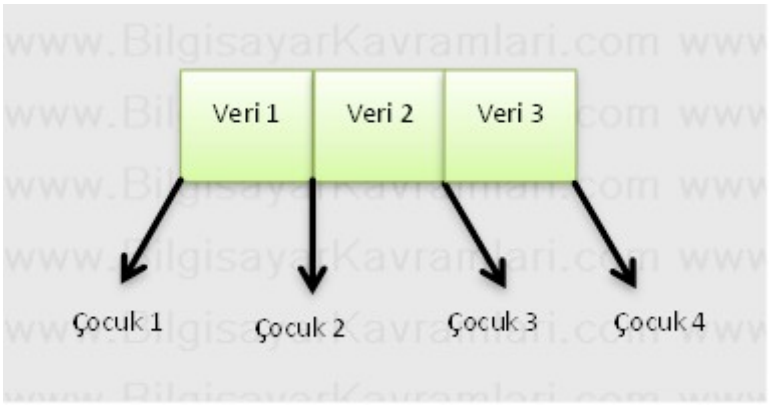
2 düğümü: 2 adet çocuk düğümü gösteren gösterici (pointer) ve bir veriden oluşur



3 düğümü: 3 adet çocuk düğümü gösteren gösterici (pointer) ve iki veriden oluşur



4 düğümü: 4 adet çocuk düğümü gösteren gösterici (pointer) ve üç veriden oluşur

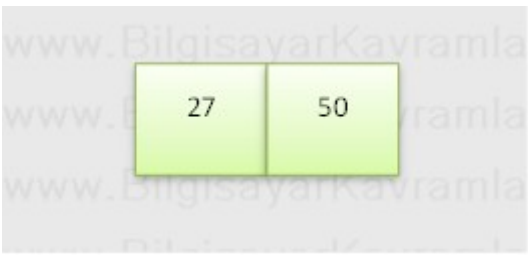


Ekleme işlemini bir örnek üzerinden anlamaya çalışalım

50, 27, 97, 52, 19, 11, 111

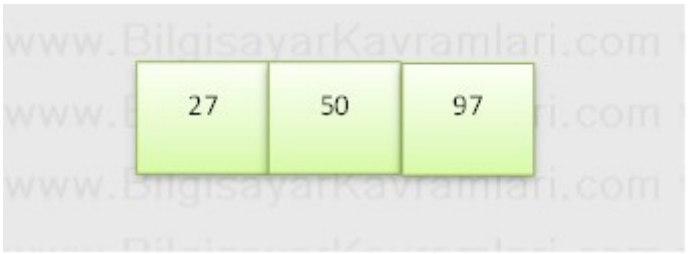
Bu sırayla verilen değerleri sırasıyla ağaca ekleyelim:

İlk sayı olan 50′nin eklenmesi sonucunda ağaçta tek bir düğüm ve bu düğümde tek bir veri bulunuyor. Bu tip düğüme 2 düğümü ismi verilmektedir ve iki çocuğu da boştur (null).

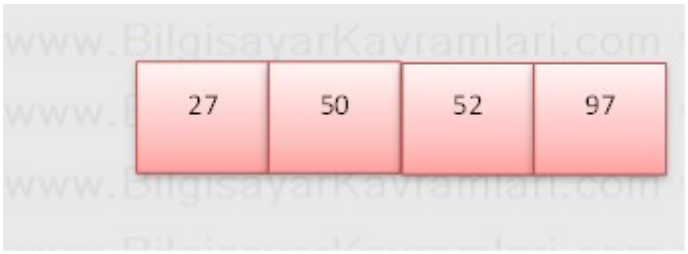


İkinci eklenen değer 27′dir. Ağacımızda tek düğüm bulunduğundan ve bu düğüm 2 tipinde olduğundan, düğümde yeni değeri alacak yer bulunmaktadır. Bu yere yeni gelen değer yerleştirilir. Elbette düğümlerin içerisinde sıra bulunması gerektiği için, 27 değeri, 50 değerinin soluna yerleştirilmiştir. Bu kural bundan sonraki ekleme işlemlerinde de geçerli olacaktır. Yani düğüm içerisindeki değerler kendi aralarında sıralı olacak ve herhangi bir değerin solundaki çocuklarının değeri, kendi değerinden küçük ve sağındaki değeri kendi değerinden büyük olacaktır.

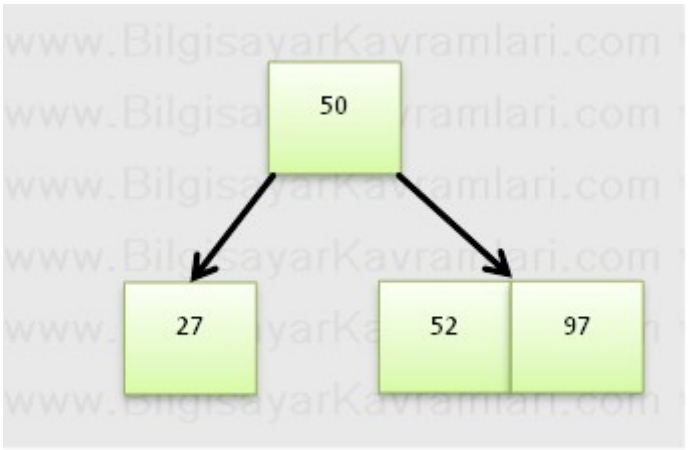
Ardından gelen ekleme değeri, 97′dir. Bu değer düğüme eklenince, aşağıdaki gibi bir yapı elde edilir:



Yukarıdaki bu yeni yapı, bizim 2-3-4 ağacımızdaki 4 tipindeki yapıdır ve kabul edilebilir bir durumdur. Ancak bir sayı daha eklenince yapı bozulacaktır. Nitekim sıradaki sayımız olan 52′nin eklendiğini düşünelim:

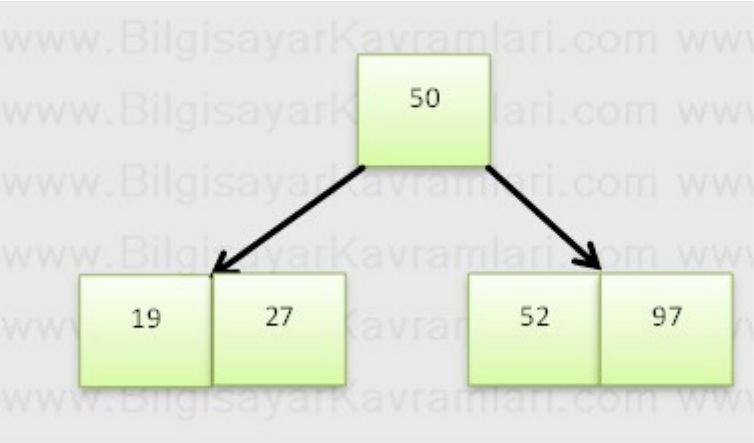


Şimdiye kadar öğrendiklerimize göre, yukarıdaki gibi bir yapı olması beklenir. Ancak yukarıdaki yapı 2-3-4 ağacı tarafından kabul edilemez (2,3 veya 4 tipinde bir düğüm değildir). Dolayısıyla bu yapının kabul edilir bir şekle indirgenmesi gerekir.



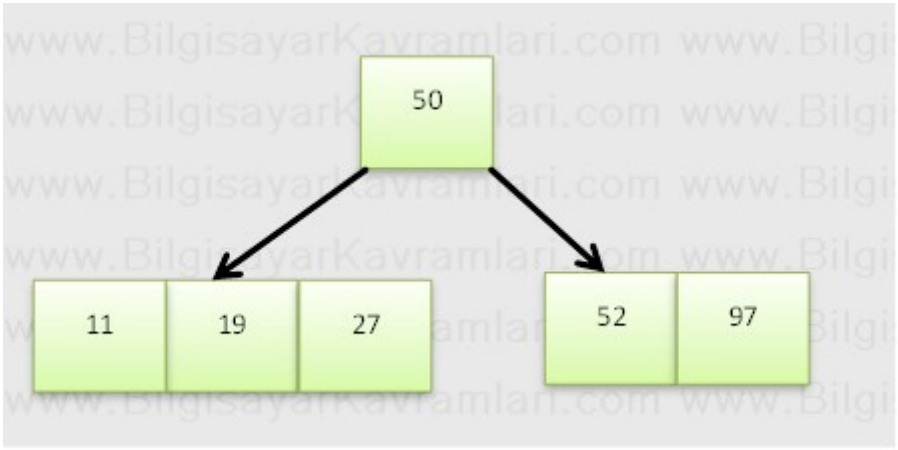
Yukarıdaki yeni şekilde, ağaç 50 ve 27 değerlerini taşıyan iki adet 2 tipinde düğüm ve bir adet 52 ve 97 değerlerini taşıyan 3 tipinde düğüme indirgenmiştir. Bu düğümlerin tamamı 2-3-4 ağacı tarafından kabul edilir düğümlerdir.

Ekleme işlemine 19 sayısı ile devam edelim.

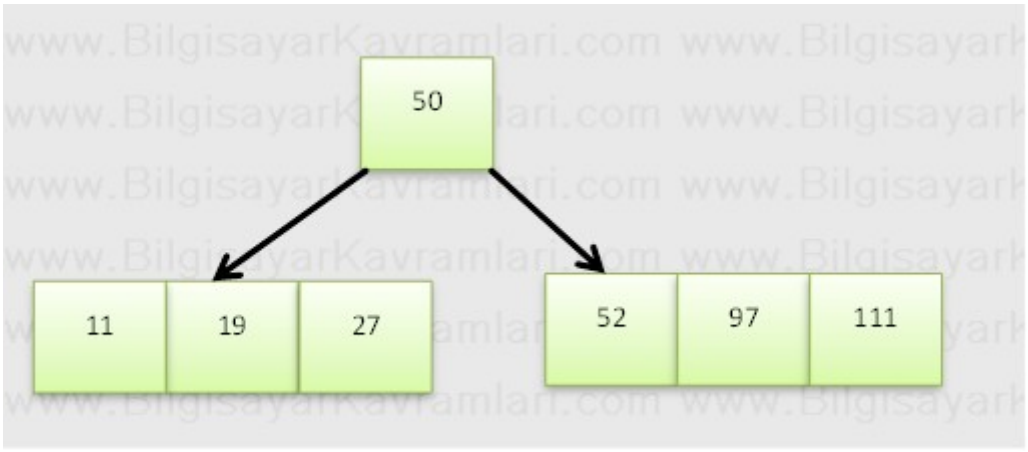


Yukarıdaki şekilde görüldüğü üzere, 19 sayısı, 50′den küçük olduğu için kökteki bu değerin solundaki düğüme yerleştirilmiştir.

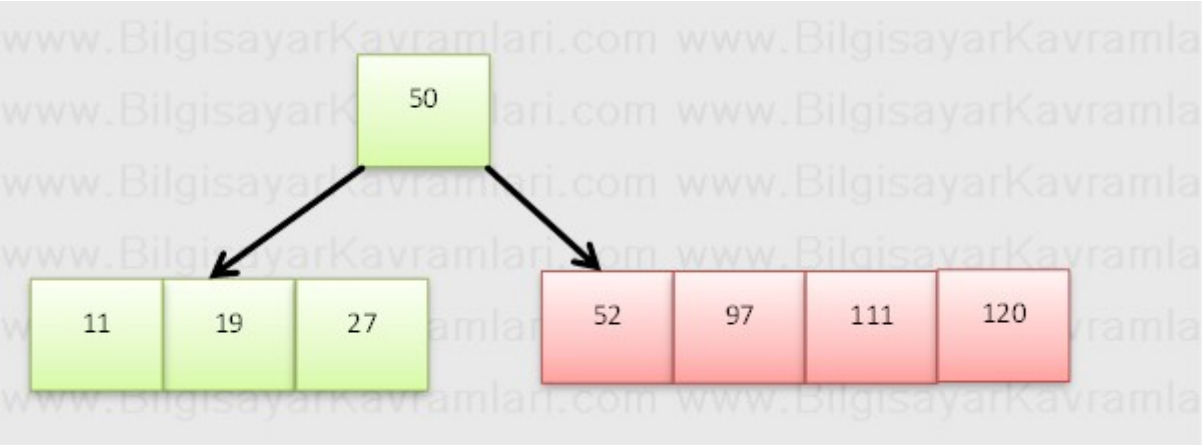
11 sayısı eklendikten sonra, bu sayı da daha önceden olduğu gibi, 50′nin solundaki düğüme sıralı olarak eklenir.



Benzer şekilde 111 sayısının eklenmesi de kök değerin sağına yapılır:

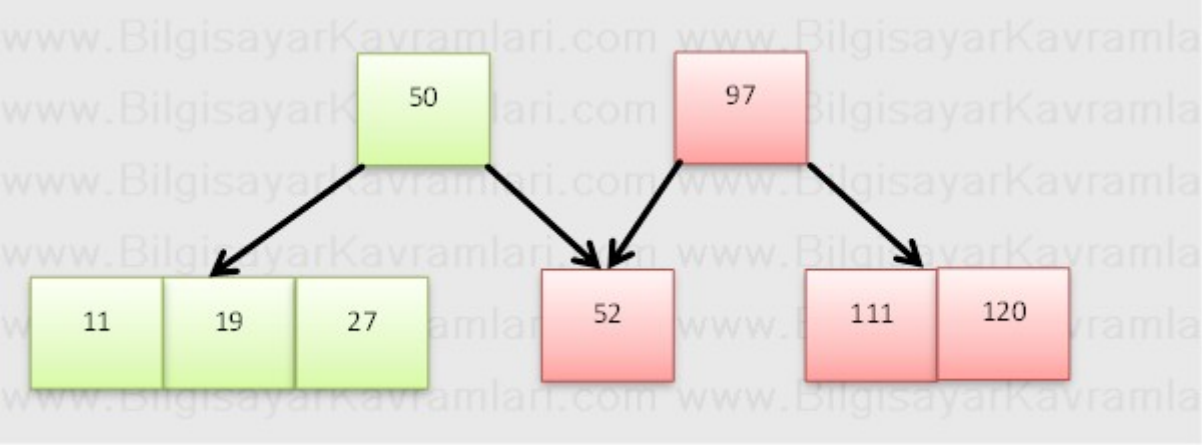


Son olarak ağacın 3 düğümü olarak taşıyamayacağı bir örneği göstermek için, örneğin 120 sayısını ağaca eklemeye çalışalım:

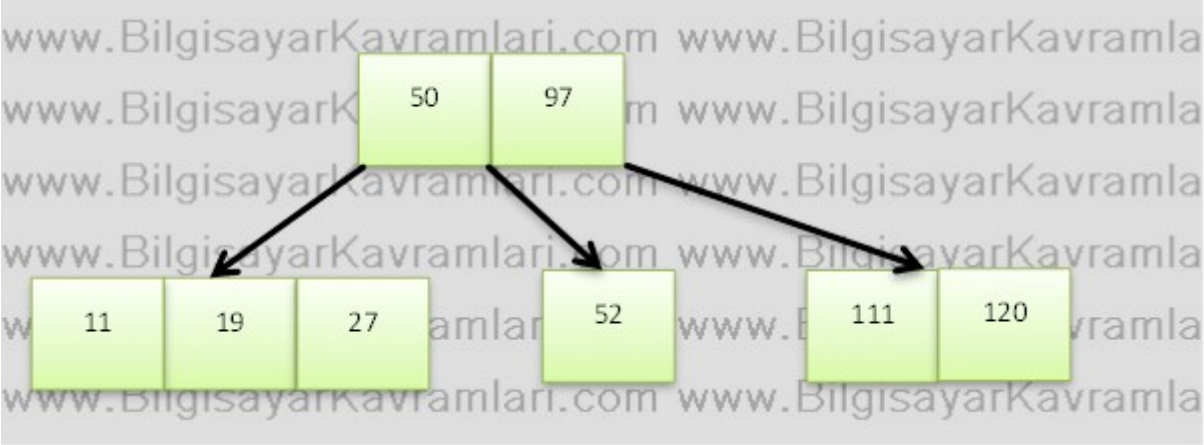


Normalde 120 sayısının ekleneceği yer, yukarıdaki şekilde gösterildiği gibi ağacın sağ koludur. Ancak bu durum 2-3-4 ağacı tarafından kabul edilemez çünkü herhangi bir düğüm yapısına uygun değildir.

Çözüm olarak daha önceden 52 sayısının eklendiği örnekte olduğu gibi düğümleri indirgiyoruz.



Bu işlemden sonra kök düğümünü güncelleyerek sonuçta çıkacak olan ağacı elde ediyoruz



Ağacımız son halinde 2-3-4 ağacı kurallarına uygun düğümlerden oluşmaktadır.

SORU-4: CCI (Computed Chaining Insertion, Hesaplamalı Zincir Eklemesi) hakkında bilgi veriniz.

Bilgisayar bilimlerinde, özellikle dosya yönetimi konusunun (file organization) kullandığı bir özetleme (hashing) çakışması (collision) çözüm algoritmasıdır. Basitçe bir özetleme fonksiyonu (hashing function) sonucunda, çalışma olması durumunda (collision), dizi üzerinde farklı bir adrese çakışan sayı konulur veya aranır. Bu farklı sayı için ikinci bir özetleme fonksiyonu kullanılır. Buraya kadar olan tanım, aslında doğrusal bölüm (linear quotient) yöntemi ile aynıdır. CCI yönteminin getirdiği en önemli farklılık ise sayıların asil yerlerinde bulunma önceliğidir. Yani bir adresleme alanında özetleme fonksiyonu sonucunda (hashing function) yerleştirilen sayılar ve çakışma sonucunda yerleştirilen sayılar bulunmaktadır. Çakışma sonucu yerleştirilen sayılar, asil yerlerinde değil de çakışma çözüm yönteminin gösterdiği yerlerdedir. CCI yöntemi, bu şekilde çakışmadan dolayı doğal adresinde olmayan sayıların bulunduğu sıraya, yeni ve asil yeri olarak bir sayı gelmesi durumunda bu yeni gelen sayının yazılmasını söyler.

Bu anlamda hesaplamalı zincir ekleme yöntemi (CCI), EISCH, LISCH veya doğrusal sondalama (linear probing) yöntemlerinin aksine hareketli bir yöntemdir (dynamic method) ve yerleşen sayıların yerlerini değiştirebilir.

Bu durumu bir örnek üzerinden anlamaya çalışalım.

Eklenecek olan sayılarımız 27, 18, 29, 28, 39, 13, 16, 38, 53, 49 olsun. Kullanacağımız özetleme fonksiyonu ise H : K mod 11 olarak verilsin.

Bu duruma mod 11 için 11 farklı alandan oluşan boş bir dizimiz bulunacaktır:

Sıra	Anahtar	Adım
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

Yukarıdaki boş tabloda, sıra sütunu, verilerin ekleneceği yeri, anahtar sütunu örnekte verilen sayıların nereye eklendiğini, adım sütunu ise, arama ve ekleme işlemleri sırasında bir sayının beklenen yerde bulunamadığında kaçar adım eklenerek aranacağıdır.

Bu anlamda CCI algoritması bağlantılı çakışma çözümü (collision with links) yapmaktadır. Yani bir çakışma durumunda, konulması gereken sıraya değil farklı bir sıraya, bir sayı konulduğunda, bu yer bağlantı bilgisinde durmaktadır.

Yukarıdaki örnekte bulunan sayıları sırasıyla ekleyelim. İlk sayımız 27 için sıra hesaplanır ve 27 mod 11 = 5 olarak bulunur ve çakışma olmadan sorunsuz bir şekilde eklenir:

Sıra	Anahtar	Adım
0		
1		
2		
3		
4		
5	27	2
6		
7		
8		
9		
10		

Yukarıdaki ekleme işlemi sırasında 27 / 2 = 2 hesaplamasına göre adım değeri de yazılır.

Ardından eklenen sayımız 18 için sıra hesaplanır ve 18 mod 11 = 7 olarak bulunur ve 18/11 = 1 bölüm değeri, adım değeri olarak çakışma olmadan sorunsuz bir şekilde eklenir:

Sıra	Anahtar	Adım
0		
1		
2		
3		
4		
5	27	2
6		
7	18	1
8		
9		
10		

Ardından eklenen sayımız 29 için sıra hesaplanır ve 29 mod 11 = 7 olarak bulunur. Yukarıdaki görüldüğü üzere 7. Sıra doludur ve dolayısıyla bir çakışma durumu söz konusudur. Çakışma durumunda, sayının eklenmesi gereken sıradaki adım değeri kadar ilerlenir ve ilk bulunan boş yere bu yeni sayı eklenir. Yukarıdaki örnekte bu adım değeri, daha önceden 7. Sıraya konulan 18 sayısından dolayı 1′dir. Dolayısıyla sayımız 1 adım atlanarak ilk boş bulunan yere yani 8. Sıraya yerleştirilir:

Sıra	Anahtar	Adım

0		
1		
2		
3		
4		
5	27	2
6		
7	18	1
8	29	
9		
10		

Sıradaki sayımı 28 mod 11 = 6. Sıraya problemsiz bir şekilde yerleşir:

Sıra	Anahtar	Adım
0		
1		
2		
3		
4		
5	27	2
6	28	2
7	18	1
8	29	
9		
10		

Bir sonraki sayı olan 39 mod 11 = 6, yerleştirilirken çakışma Çakışma olan 6. Sıradaki adım değeri 2′dir dolayısıyla yeni sayımız 39, bu adım değeri kadar atlanarak ilk boş bulunan yere yerleştirilecektir. 2 adım atlandığında 8. Sıra bulunur ve burası doludur, tekrar 2 adım atlanır ve 10. Sırada bulunan boş yere 39 sayısı yerleştirilir.:

Sıra	Anahtar	Adım
0		
1		
2		
3		
4		
5	27	2
6	28	2
7	18	1
8	29	
9		
10	39	

13 sayısı sorunsuz bir şekilde 13 mod 11 = 2. Sıraya yerleştirilir:

Sıra	Anahtar	Adım
0		
1		
2	13	1
3		
4		
5	27	2
6	28	2
7	18	1
8	29	
9		
10	39	

16 sayısı ise 16mod11= 5. Sıra dolu olduğu için çakışır. 5. Sıradaki adım değeri 2′dir dolayısıyla 2 adım atlanarak ilk boş bulunan yere sayı yerleştirilecektir. 7. sıra doludur dolayısıyla 9. Sıraya yerleştirilir.

Sıra	Anahtar	Adım
0		
1		
2	13	1
3		
4		
5	27	2
6	28	2
7	18	1
8	29	
9	16	
10	39	

38 sayısı da benzer şekilde problemli bir sayıdır. 38 mod 11 = 5 sırası doludur. Çözüm olarak 5. Sıradaki adım miktarı kadar attırılarak boş yer aranır ve 5, 7, 9, 0. Sıralara bakılarak , boş bulunan 0. Sıraya yerleştirilir.

Sıra	Anahtar	Adım
0	38	
1		
2	13	1
3		
4		
5	27	2
6	28	2
7	18	1

8	29	
9	16	
10	39	

Ardından gelen 53 sayısı için benzer şekilde çakışma olur çünkü 53 mod 11 = 9. Sıra doludur. Bu sayının, şimdiye kadar olan sayılardan farkı, çakıştığı ve daha önceden 9. Sıraya yerleşen sayı olan 16 sayısının doğal yerinde bulunmamasıdır. Yani 9. Sırada, daha önceden yerleşen 16 sayısı, aslında 9. Sıraya yerleşmesi gerekmeyen ancak 5. Sıra dolu olduğu için buraya yerleştirilmiş sayıdır.

Bu durumda, CCI algoritması, doğal olarak yerleşen sayıya, çakışmadan dolayı yerleştirilen sayıdan öncelik tanımaktadır. 16 sayısının yerine 53 sayısı öncelikli olarak yerleştirilir.

Sıra	Anahtar	Adım
0	~~38~~ 16	1
1	38	
2	13	1
3		
4		
5	27	2
6	28	2
7	18	1
8	29	
9	~~16~~ 53	1
10	39	

53 sayısının yerleştirilmesi üzerine 16 sayısı sıradaki adım değeri olan ve 1 adım sonrası olan 0. Sıraya taşınır. Burada bulunan 38 sayısı da bir sonraki boş sıraya taşınarak adım değerleri buna göre güncellenir.

Sıra	Anahtar	Adım
0	16	1
1	38	
2	13	1
3		
4		
5	27	2
6	28	2
7	18	1
8	29	
9	53	1
10	39	

Son olarak gelen 49 sayısı, yerleşmesi gereken 49 mod 11 = 5. Sıra dolu olduğu için 5. Sırada bulunan adım değeri kadar atlanarak boş yer aranır. Sırayla, 5, 7, 9,0,2 ve 4.. Sıralara bakılır. 4. Sırada boş yer bulunduğu için buraya yerleştirilir:

Sıra	Anahtar	Adım
0	16	1
1	38	
2	13	1
3		
4	49	
5	27	2
6	28	2
7	18	1
8	29	
9	53	1
10	39	

Yukarıdaki örnekte sayıların nasıl yerleştirildiğini gördük. Bu yöntemi özetleyecek olursak:

1. Yerleşecek sayının (anahtarın) özetleme fonksiyonu değerini (hashing function) hesapla ayrıca bölümden gelen adım değerini de hesaplayarak buraya yerleştir.
2. Şayet burası doluysa, hesaplanan adım değeri kadar atlanarak boş olan ilk yere sayı yerleştirilir.
3. Şayet sayının yerleşeceği yer doluysa ve buradaki sayı, normalde yerleşmesi gereken yer değilse o zaman yeni gelen sayı önceliğe sahip olup buradaki sayı ile yer değiştirilir.

Yukarıdaki bu 3 basit adım takip edilerek CCI algoritmasına göre yerleştirme işlemi yapılabilir.

brent yöntemine (Brent's method) benzer şekilde, CCI yöntemi de sınıflandırma bakımından, hareketli metot (dynamic method) olarak kabul edilebilir. Bundan kasıt, bir sayı bir sıraya yerleştikten sonra buradan hareket edebilir, bu durum yukarıdaki örnekte 53 sayısının yerleştirilmesi sırasında açıkça görülmektedir.

SORU-5: RAID (Redundant Array of Independent Disks) hakkında bilgi veriniz.

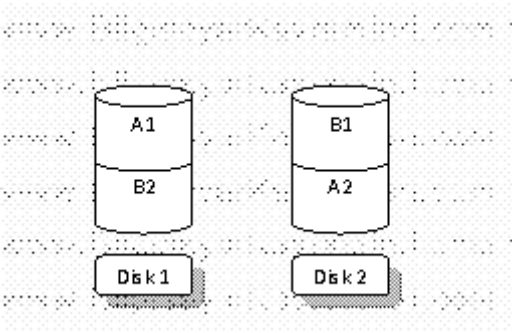
Bilgisayar bilimlerinde, depolama yönetimi (storage management) için kullanılan yöntem standardıdır. Kaynaklarda redudant array of independent disks (fazladan bağımsız disk dizisi) şeklinde geçtiği gibi , redundant array of inexpensive disks (ucuz disklerin fazlalık dizisi) olarak da geçmektedir.

Kısaca bu standart çeşitli seviyelerde verinin fazladan bir kopyasının tutulması ile birlikte herhangi bir problem durumunda veri kaybını engellemeyi hedefler.

RAID – 0′dan RAID – 6′ya kadar seviyeleri bulunmaktadır. Bu yazıda en çok kullanılan RAID0, RAID1 , RAID3 , RAID5 ve RAID6 seviyeleri açıklanacak ayrıca RAID10 ve RAID01 yapıları gösterilecektir. Geriye kalan RAID2 ve RAID4 seviyeleri güncel hayatta çok kullanılmadığı için bu yazı kapsamı dışında bırakılmıştır.

RAID0

Bu seviyede veriler birden fazla diske (klasik tanımımızda en az 2 disk olarak geçer) dağıtılmaktadır. Buradaki amaç, veriye erişim süresini kısaltmak ve bir problem, bir müşkülat halinde verinin tamamının kaybını engellemektir. Bu seviyede, müşkülat durumunda veri kaybı muhtemeldir.

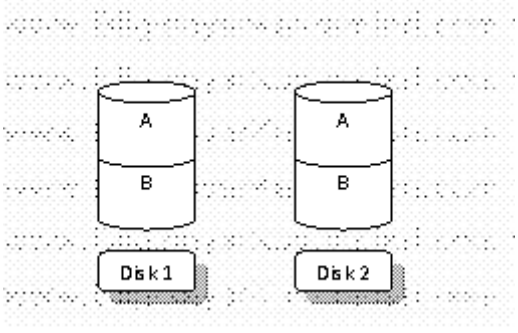


Yukarıdaki şekilde görüldüğü üzere iki ayrı diske iki ayrı veri dağıtılmıştır. A verisi iki parçaya bölünmüş ve bir kısmı 1. Diske diğer kısmı ise ikinci diske yazılmıştır. Disklerden birisinin kaybedilmesi durumunda hiçbir bilgi tamamen kaybedilmez. Ayrıca A verisinin okunması durumunda iki disk paralel olarak çalışarak veriyi daha hızlı okuyabilir.

RAID0 seviyesine kaynaklarda izli veya çizgili disk anlamına gelen striped disk ismi verilmektedir. Ayrıca bazı kaynaklarda RAID0, klasik raid seviyelendirmesi içerisinde sayılmaz. Bunun sebebi herhangi bir şekilde fazladan tutulan (raid kelimesinin açılımında redundant – fazladan olduğunu hatırlayınız) disk bulunmamaktadır. Verilerin tutulması için gereken en az seviyede disk harcanmış olunur.

RAID1

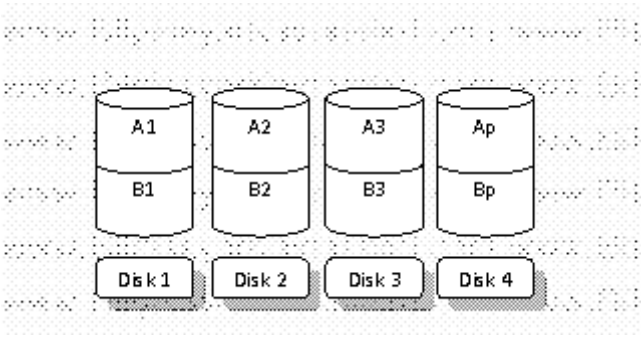
Bu seviye literatürde aynalama (mirroring) olarak geçer. Burada ifade edilmek istenen, bir diskteki her şeyin bir yansımasını (bir kopyasını) ikinci bir diskte tutmaktır. Dolayısıyla disklerden birisine bir şey olması durumunda diğerine bulunan kopya sayesinde veri kurtulabilir. Bu yöntem hız açısından bir avantaj sağlamaz, sebebi disklerin ikisinin paralel olarak aynı veriyi yazmaları ve okunurken tek bir diskten okunmasıdır. Verinin korunması açısından da raid teknolojileri arasındaki en maliyetli yöntemdir çünkü aktif olarak kullanılan depolama alanının 2 misli yer işgal edilir. Örneğin 100GB veri saklamak için aslında 200GB alan işgali söz konusudur.



Yukarıdaki şekilde görüldüğü üzere iki disk de birbirinin kopyasıdır.

RAID3

Bu seviyeden sonraki RAID yöntemlerinde verinin çiftlik kontrolünden (parity check) istifade edilerek ilave bir disk bu verinin saklanması için kullanılır.



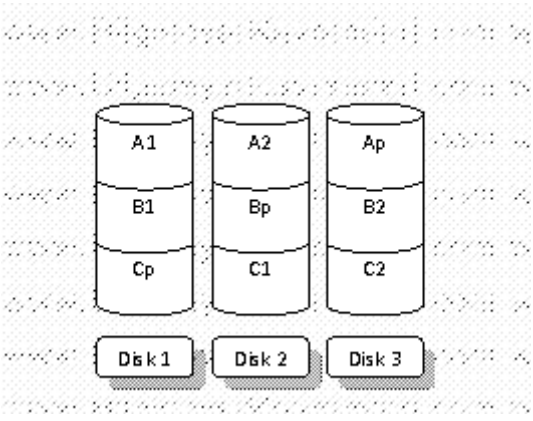
Yukarıdaki şekilde görüldüğü üzere 4 diskten bir tanesi bu kontrol bitlerini tutmakta (parity) ve dolayısıyla normalde kullanılan veriye ilave bir disk gerekmektedir. Yukarıdaki gösterime benzer bir kurulumda disklerden birisinin bozulması veya bu diskteki verinin kaybedilmesi halinde diğer 3 diskte bu diskte bulunan veri geri üretilebilir.

Hız açısından bu kurulum incelendiğinde de verinin 3 ayrı disk üzerine paralel olarak yazılması veya okunması, şimdiye kadar olan seviyelere göre belirgin bir hız artışı sağlar. Ne yazık ki bir dezavantajı partiy bitlerinin yazıldığı 4. Diske de veri yazılıyor/okunuyor olmasıdır ve bu da ilave veri iletişimi ve vakit kaybı anlamına gelir. Yine de bu zaman kaybı şimdiye kadar olan diğer yöntemlere göre göz ardı edilebilir seviyededir.

RAID3 kurulumu için en az 3 disk gerekmektedir. Bu durumda 2 disk veriyi tutmakta ve 3. Disk kontrol amaçlı kullanılmaktadır. O halde 2/3 = %66 oranında alan aktif veri saklamak için kullanılır denilebilir. Ancak kontrol diskine ilave 5 disk kullanılması durumunda bu aktif alan 5/6 = %86 gibi bir orana çıkmaktadır.

RAID 5

Bu seviye, raid 3 benzeri bir şekilde verinin parity kontrolünü çıkarmakta ve ilave bir diskte tutmaktadır. Ancak verinin kendisini ve parity biti farklı disklere dağıtmaktadır. Bu yüzden herhangi bir diskte problem olması durumunda sistem kendisini kurtarana ve bu problemli diski düzeltene kadar çalışmada aksama olur.

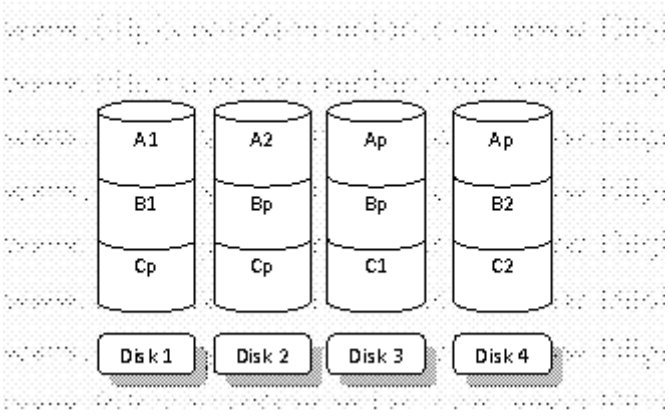


Şekilde görüldüğü üzere her disk, farklı bir bilginin parity kontrolünü tutmakta dolayısıyla disklerden birisinde bir müşkülat olması durumunda diğerlerindeki veri veya parity ile bu

sorunlu disk kurtulabilmektedir. Bu yöntemde tek diskin kaybedilmesi durumunda veri kurtarılabilirken ikinci bir disk kaybedilirse veri kaybı oluşur.

RAID6

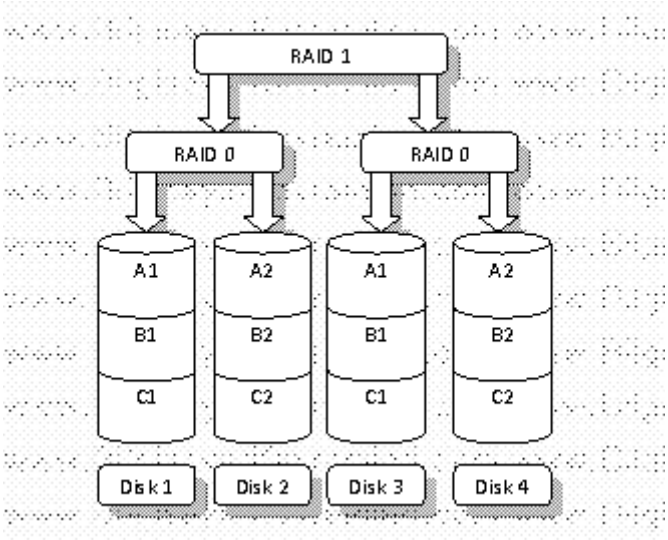
Şimdiye kadar olan yöntemlerden farklı olarak parity bitlerin iki kopyası tutulur. Bu durum aşağıdaki resimde gösterilmiştir:



Bu yeni dizilimin kazandırdığı avantaj, iki diskin birden kaybedilmesi durumunda verinin kurtarılabilmesidir.

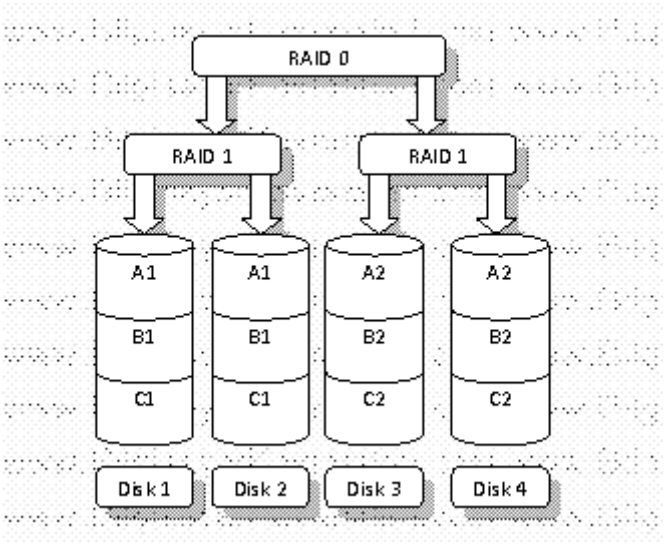
RAID 10 ve RAID 01

Bu iki raid yapısı, aynalama (mirroring) kullanırken bir ağaç yapısına benzer dağılım izler. Örneğin 4 diski aynalamak istiyorsak toplamda 8 disk olacaktır ve bu 8 diskin nasıl dağılabileceği aşağıdaki iki örnekte gösterilmiştir:



Yukarıdaki gösterilen raid01 yapısıdır ve diskler önce raid0 yapısına göre raidlenmiş ardından da 1 yapısına göre aynalanmıştır. Yani yukarıda anlatılan raid 0 yapısı hatırlanacak olursa, veriler diskler üzerinde dağıtılmıştır. Ardından raid 1 yapısı ile bu yapılar kopyalanmıştır ve iki kopyası tutulmaktadır. Diğer bir deyişle disk3 ve disk4 arasında raid 0 ve disk 3 ve disk 1 arasında raid 1 uygulanmıştır.

Bu uygulama tersten uygulanırsa aşağıdaki şekilde bir görüntü çıkar:



Bu yeni yapıda ise diskler önce kendi aralarında raid 1 ile aynalanıp (mirroring) ardında raid 0 ile bu aynalanmış gruplar bağlanır.

RAID 01 ve RAID 10 kurulumları karmaşıklığı arttırmasına karşılık yüksek miktardaki disk yönetiminde tercih edilmektedir.

SORU-6: Doğrusal Bölüm (Linear Quotient) hakkında bilgi veriniz.

Bilgisayar bilimlerinde, özellikle dosya yönetimi konusunun (file organization) kullandığı bir özetleme (hashing) çakışması (collision) çözüm algoritmasıdır. Basitçe bir çakışma durumu olduğunda, eklenecek olan anahtarı kaç sıra sonraya yerleştireceğimizi bulan ikinci bir özetleme fonksiyonu kullanılır. Kullanılan ikinci özetleme fonksiyonu ise sayının bölümüdür:

H1 : K mod n

H2 : K / n

Olarak düşünülebilir. Bu anlamda çift özetleme fonksiyonlarına (double hashing) benzetilebilir.

Yöntemimizin çalışmasını bir örnek üzerinden anlamaya çalışalım.

Eklenecek olan sayılarımız 27, 18, 29, 28, 39, 13, 16, 42, 17 olsun. Kullanacağımız özetleme fonksiyonu ise H1 : K mod 11 ve H2 : K / 11 olarak verilsin.

Bu duruma mod 11 için 11 farklı alandan oluşan boş bir dizimiz bulunacaktır:

Sıra	Anahtar
0	
1	
2	

3	
4	
5	
6	
7	
8	
9	
10	

Yukarıdaki boş tabloda, sıra sütunu, verilerin ekleneceği yeri, anahtar sütunu örnekte verilen sayıların nereye eklendiğini tutmaktadır. Diğer çakışma çözüm yöntemleri olan LISCH yönteminin veya EISCH yönteminin tersine bu yöntemde bir bağlantı sütunu bulunmaz. Bunun sebebi çakışma durumunda sayının konulacağı veya aranacağı adresin ikinci bir özetleme fonksiyonu (hashing function) ile hesaplanabiliyor olmasıdır.

Bu anlamda doğrusal bölüm (linear quotient) algoritması bir doğrusal sondalama (linear progbing veya progressive overflow) olarak düşünülebilir ve bağlantısız çakışma çözümü (collision without links) olarak sınıflandırılabilir.

Yukarıdaki örnekte bulunan sayıları sırasıyla ekleyelim. İlk sayımız 27 için sıra hesaplanır ve 27 mod 11 = 5 olarak bulunur ve çakışma olmadan sorunsuz bir şekilde eklenir:

Sıra	Anahtar
0	
1	
2	
3	
4	
5	27
6	
7	
8	
9	
10	

Ardından eklenen sayımız 18 için sıra hesaplanır ve 18 mod 11 = 7 olarak bulunur ve çakışma olmadan sorunsuz bir şekilde eklenir:

Sıra	Anahtar
0	
1	
2	
3	
4	

5	27
6	
7	18
8	
9	
10	

Ardından eklenen sayımız 29 için sıra hesaplanır ve 29 mod 11 = 7 olarak bulunur. Yukarıdaki görüldüğü üzere 7. Sıra doludur ve dolayısıyla bir çakışma durumu söz konusudur. Çakışma durumunda ikinci özetleme fonksiyonumuz devreye girer ve 29 / 11 = 2 bulunur. Bu durumda sayının konulması gereken yerden 2 sonraki adrese koymayı denememiz gerekir. Örneğimizde 7. Sıraya konması gereken sayımızın ikinci özetleme fonksiyonu sonucu 2′dir ve dolayısıyla 9. Sıraya koymaya çalışırız. Henüz boş olan bu sıraya sayı başarı ile yerleştirilir.

Sıra	Anahtar
0	
1	
2	
3	
4	
5	27
6	
7	18
8	
9	29
10	

Sıradaki sayımı 28 mod 11 = 6. Sıraya problemsiz bir şekilde yerleşir:

Sıra	Anahtar
0	
1	
2	
3	
4	
5	27
6	28
7	18
8	
9	29
10	

Bir sonraki sayı olan 39 mod 11 = 6, yerleştirilirken çakışma olur, dolayısıyla sayının ikinci özetleme fonksiyonu değeri hesaplanır. Bu değer 39 / 11 = 3 olarak bulunur ve normalde konulması gereken 6. Sıradan 3 sıra sonraya yani 9. Sıraya yerleştirilmeye çalışılır. Ancak bu adres de doludur. Çözüm olarak 3 sıra daha ilerlenir ve 12. Sıraya (mod 11′de çalıştığımız için 1. Sıra oluyor, okuyucu bunu sona ulaşıldığında baştan devam edilir şeklinde de düşünebilir) yerleştirilir:

Sıra	Anahtar
0	
1	39
2	
3	
4	
5	27
6	28
7	18
8	
9	29
10	

13 sayısı sorunsuz bir şekilde 13 mod 11 = 2. Sıraya yerleştirilir:

Sıra	Anahtar
0	
1	39
2	13
3	
4	
5	27
6	28
7	18
8	
9	29
10	

16 sayısı ise 16mod11= 5. Sıra dolu olduğu için çakışır. 16/11 = 1 olduğu için birer eklenerek ilk boş yer bulunana kadar arama yapılır. Sırasıyla 6. 7. Ve 8. Sıralara bakılır. Nihayet 8. Sırada boşluk bulunduğu için sayı buraya yerleştirilir:

Sıra	Anahtar
0	
1	39
2	13

3	
4	
5	27
6	28
7	18
8	16
9	29
10	

42 sayısı da benzer şekilde problemli bir sayıdır. 42 mod 11 = 9. Sıra doludur. İkinci özetleme fonksiyonumuza göre 42 / 11 = 3′tür ve yerleştirme 9 + 3 = 12 mod 11 = 1 sırasına yapılmaya çalışılır mamafih bu alanda dolu olduğu için bir kere daha 3 eklenerek 1 + 3 mod 11 = 4. Sıraya yerleştirilir:

Sıra	Anahtar
0	
1	39
2	13
3	
4	42
5	27
6	28
7	18
8	16
9	29
10	

Son olarak gelen 17 sayısı, yerleşmesi gereken 17 mod 11 = 6. Sıra dolu olduğu için 17/11 = 1 değerine göre birer arttırılarak boş bir yer aranır. Sırasıyla 7. 8. 9. Ve 10. Sıralara bakılır. Boş sıra bulununca 17 sayısı buraya yerleştirilir:

Sıra	Anahtar
0	
1	39
2	13
3	
4	42
5	27
6	28
7	18
8	16
9	29
10	17

Yukarıdaki örnekte sayıların nasıl yerleştirildiğini gördük. Bu yöntemi özetleyecek olursak:

1. Yerleşecek sayının (anahtarın) özetleme fonksiyonu değerini (hashing function) hesapla ve buraya yerleştir.
2. Şayet burası doluysa, ikinci özetleme fonksiyonuna, yani bölme işlemine tabi tutup bulunan değer kadar sonrasına bak.
3. Boş alan bulunana kadar bölme işlemi sonucunu ekleyerek arama yap.

Yukarıdaki bu 3 basit adım takip edilerek doğrusal bölüm (linear quotient) yöntemine göre yerleştirme işlemi yapılabilir.

Yerleşen bu sayıların aranması ise oldukça basittir.

Örneğin aranan sayının 42 olduğunu düşünelim. 42 mod 11 = 9 olarak bulunur ve 9. Sırada bu sayı aranır. Sayı burada olmadığı için aranan sayının ikinci özetleme fonksiyonu hesaplanır. 42 / 11 = 3′tür ve dolayısıyla 9. Sıradan başlanarak 3′er arttırılarak bu sayı aranır. 9 + 3 = 12 mod 11 = 1. Sıraya bakılır ancak sayı yoktur, tekrar 3 arttırılır ve 4. Sırada sayı bulunur.

Benzer şekilde, dizide hiç bulunmayan bir sayı aranırsa, arama işlemine başladığımız sıraya gelinmesi veya bütün diziye bakılması durumunda işlem iptal edilir. Örneğin dizimizde olmayan 34 sayısını aramak istediğimizi düşünelim.

34 mod 11 = 3 bulunup bu sıraya bakılacak, bu sırada hiç sayı olmadığı için arama işlemi sayının dizide olmadığını rahatlıkla söyleyebilecektir.

Aranan sayımız 48 olsaydı. 48 mod 11 = 4 bulunacak ve 4. Sırada sayı aranacaktır. 4. Sırada sayı olmadığı için 48 / 11 = 4 arttırılarak arama işlemi devam edecektir.

Sırasıyla 8, 1,5,9,2,6,10,3,7,0 sıralarına bakılacak ve en sonunda bakılacak değer 4 olarak hesaplanınca arama işlemi nihayete erdirilecektir.

Bu durumda da görüldüğü üzere, olmayan bir sayının aranması bazı durumlarda bütün indeksleme alanına bakılmasını gerektirebilir. Bu performans açısından kötü bir durumdur.

Arama algoritmasını aşağıdaki şekilde yazabiliriz:

1. Sayının özetleme fonksiyonu (hashing function) değerini hesapla ve sayıyı bu bulduğun sırada ara
2. Sayı burada ise sayıyı buldun. Aramayı bitir.
3. Sayı yoksa 2. Özetleme değeri olan bölümü hesapla.
4. Baktığın adresi bölüm değeri kadar arttırarak tekrar bak.
5. Sayıyı bulduysan bitir.
6. Başladığın adrese döndüysen aramayı bitir ve sayının bulunmadığını söyle.
7. 4. Adıma geri git.

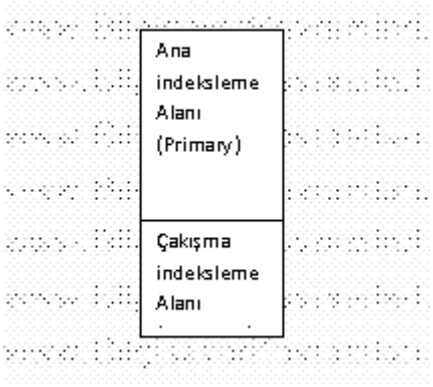
Yukarıda anlatılan doğrusal bölüm yöntemi, doğrusal sondalama (linear probing) çatısı altında kabul edilebilir. Bu tip yöntemlerde, EISCH yöntemi veya LISCH yönteminde bulunan bağlantı sütunları (links) kullanılmaz. Bunun bir neticesi olarak arama sırasında bulunmayan bir sayı için bütün arama alanına bakılması gerekebilir. Bu durum bir dezavantajdır ancak bir avantajı, bağlantı sütunu tutulmadığı için yer kazanılmış olunur.

Doğrusal bölüm (linear quotient) yöntemi yine sınıflandırma bakımından sabit metot (static method) olarak kabul edilebilir. Bundan kasıt, bir sayı bir sıraya yerleştikten sonra buradan hareket edemez, ilk defa yerleştiği yerde kalır.

SORU-7: LICH (Last Insertion Coalesced Hashing) hakkında bilgi veriniz.

Türkçeye, son ekleme birleştirme özetlemesi olarak çevrilebilir. Bilgisayar bilimlerinde, özellikle dosya yönetimi konusunun (file organization) kullandığı bir özetleme (hashing) çakışması (collision) çözüm algoritmasıdır. Basitçe bir özetleme alanını iki parçaya ayıran bu algoritmada amaç çakışma sonucu oluşan alanlar ile doğru adreste indekslenen verilerin ayrılmasıdır.

Kabaca aşağıdaki şekil gibi düşünebiliriz:



Yukarıdaki şekilde görüldüğü üzere, indeksleme alanı ana alan (primary) ve çakışma alanı (overflow) olarak ikiye bölünmüş ve dolayısıyla çakışmadan dolayı farklı bir sıraya yerleşen sayılar ile olması gereken yerde bulunan sayılar birbirinden ayrılmıştır.

Yöntemimiz, bir özetleme fonksiyonu (hashing function) sonucunda, çalışma olması durumunda (collision), dizinin sonundan başa doğru boş bulunan ilk yere yerleştirmeyi söyler.

Bu durumu bir örnek üzerinden anlamaya çalışalım.

Eklenecek olan sayılarımız 27, 18, 29, 28, 39, 13, 16, 42, 17 olsun. Kullanacağımız özetleme fonksiyonu ise H : K mod 7 olarak verilsin.

Bu duruma mod 7 için 7 farklı alandan oluşan boş bir dizimiz bulunacaktır. Bu 7 sıra içeren ana indeksleme alanının yanında bir de çakışma alanımız (overflow) bulunmalıdır. Bu alanı da problemimizde 4 olarak tanımlayalım:

Sıra	Anahtar	Bağlantı
0		
1		
2		
3		
4		
5		

6		
7		
8		
9		
10		

Yukarıdaki boş tabloda, sıra sütunu, verilerin ekleneceği yeri, anahtar sütunu örnekte verilen sayıların nereye eklendiğini, bağlantı sütunu ise, arama ve ekleme işlemleri sırasında bir sayının beklenen yerde bulunamadığında nerede aranacağı bilgisini tutmaktadır. Ayrıca dikkat edilirse 6. Ve 7. Sıralar arasında da bir çizgi ile ana indeksleme ve çakışma indeksleme alanları ayrılmıştır. Amacımız çakışarak farklı yere yerleştirilen sayılar ile doğal indeksleme sırasında bulunan sayıları ayırmaktır.

Yukarıdaki örnekte bulunan sayıları sırasıyla ekleyelim. İlk sayımız 27 için sıra hesaplanır ve 27 mod 7 = 6 olarak bulunur ve çakışma olmadan sorunsuz bir şekilde eklenir:

Sıra	Anahtar	Bağlantı
0		
1		
2		
3		
4		
5		
6	27	
7		
8		
9		
10		

Ardından eklenen sayımız 18 için sıra hesaplanır ve 18 mod 7 = 4 olarak bulunur ve çakışma olmadan sorunsuz bir şekilde eklenir:

Sıra	Anahtar	Bağlantı
0		
1		
2		
3		
4	18	
5		
6	27	
7		
8		
9		
10		

Ardından eklenen sayımız 29 için sıra hesaplanır ve 29 mod 7 = 1 olarak bulunur ve çakışma olmadan sorunsuz bir şekilde eklenir:

Sıra	Anahtar	Bağlantı
0		
1	29	
2		
3		
4	18	
5		
6	27	
7		
8		
9		
10		

Sıradaki sayımı 28 mod 7 = 0. Sıraya problemsiz bir şekilde yerleşir:

Sıra	Anahtar	Bağlantı
0	28	
1	29	
2		
3		
4	18	
5		
6	27	
7		
8		
9		
10		

Sıradaki sayımı 39 mod 7 = 4. Sıraya yerleşmesi gerekir. Ancak 4. Sırada daha önceden yerleştirilen 18 sayısı bulunmaktadır. Bu durumda sayı çakışma durumunda olduğu üzere çakışma alanındaki sondan boş olan ilk yere yerleştirilir. Çakışma alanımız 7 ile 10. Sıralar arasıdır ve şu anda bütün sıralar boştur. Bu sıralardan en altta olan 10. Sıraya çakışan sayımızı yerleştiriyoruz. Sayımızın normalde olması gereken 4. Sıraya da 10. Sıraya bir çakışan sayı yerleştirdiğimiz için bağlantı ekliyoruz ve 4′ten 10. Sıraya aşağıda görüldüğü üzere ilerideki aramalar için bir bağlantı kuruluyor:

Sıra	Anahtar	Bağlantı
0	28	
1	29	
2		
3		
4	18	10

5		
6	27	
7		
8		
9		
10	39	

Sıradaki sayımı 13 mod 7 = 6. Sıraya yerleşmesi gerekirken yine bir çakışma ile karşılaşıyoruz. Çakışan sayılar için ayrılan alt alanımızdaki en alttaki en boş sıra 9 ve çakışan sayı buraya yerleştiriliyor, ilave olarak bir bağlantı ile sayının 6. Sırada olması gerekirken yerleştirildiği bu çakışma sırası bağlanıyor:

Sıra	Anahtar	Bağlantı
0	28	
1	29	
2		
3		
4	18	10
5		
6	27	9
7		
8		
9	13	
10	39	

Sıradaki sayımı 16 mod 7 = 2. Sıraya problemsiz bir şekilde yerleşir:

Sıra	Anahtar	Bağlantı
0	28	
1	29	
2	16	
3		
4	18	10
5		
6	27	9
7		
8		
9	13	
10	39	

Sıradaki sayımı 42 mod 7 = 0. Sıraya yerleşmesi gerekirken yine bir çakışma ile karşılaşıyoruz. Çakışan sayılar için ayrılan alt alanımızdaki en alttaki en boş sıra 8 ve çakışan sayı buraya yerleştiriliyor, ilave olarak bir bağlantı ile sayının 0. Sırada olması gerekirken yerleştirildiği bu çakışma sırası bağlanıyor:

Sıra	Anahtar	Bağlantı
0	28	8
1	29	
2	16	
3		
4	18	10
5		
6	27	9
7		
8	42	
9	13	
10	39	

Sıradaki sayımı 17 mod 7 = 3. Sıraya problemsiz bir şekilde yerleşir:

Sıra	Anahtar	Bağlantı
0	28	8
1	29	
2	16	
3	17	
4	18	10
5		
6	27	9
7		
8	42	
9	13	
10	39	

Yukarıdaki örnekte görüldüğü üzere özetleme fonksiyonu sonucunda çakışan sayılar, çakışma alanına (overflow) yerleşirken 0 ile 6. Sıralar arasında belirlenen ana alanda sadece özetleme fonksiyonundan gelen asil sayılar durmaktadır. Bu durumu EISCH yöntemi ile karşılaşırsak, sayıları düzenli bir şekilde yerleştirmesi LICH yönteminin bir avantajıdır. Ancak Ayrılan çakışma alanı veya ana alanın dolması durumunda diğer alanı kullanamıyor olması LICH yönteminin yer israfı yapmasına sebep olur. Bu ise bir dezavantaj olarak görülebilir. Yani yukarıdaki örnekte, 2 adet çakışma daha olsaydı, çakışma alanında tek yer olduğu için 2. Çakışmayı yerleştiremeyecek özetleme alanımız dolu hatasını verecektik. Oysaki EISCH yöntemi burada boş olan herhangi bir yeri kullanabilmektedir.

Yukarıdaki örnekte sayıların nasıl yerleştirildiğini gördük. Bu yöntemi özetleyecek olursak:

1. Yerleşecek sayının (anahtarın) özetleme fonksiyonu değerini (hashing function) hesapla ve buraya yerleştir.
2. Şayet burası doluysa, dizideki çakışma alanında (overflow) bulunan boş en alt hücreye yerleştir ve bu yerleştirdiğin yere, normalde yerleşmesi gereken yerden bir bağlantı kur

Yukarıdaki bu 2 basit adım takip edilerek LICH algoritmasına göre yerleştirme işlemi yapılabilir.

Yerleşen bu sayıların aranması ise oldukça basittir.

Örneğin aranan sayının 13 olduğunu düşünelim. 13 mod 7 = 6 olarak bulunur ve 6. Sırada bu sayı aranır. Sayı burada olmadığı için buradaki bağlantı olan 9. Sıraya gidilip bakılır ve 13 sayısı bulunmuş olunur.

Benzer şekilde, dizide hiç bulunmayan bir sayı aranırsa, bağlantısı olmayan bir sıraya gelindiğinde bu sayının dizide olmadığına kanaat getirip durulabilir. Örneğin dizimizde olmayan 34 sayısını aramak istediğimizi düşünelim.

34 mod 7 = 6 bulunup bu sıraya bakılacak, bu sırada 34 olmadığı görülünce bağlantının gösterdiği 9. Sıraya bakılacak ve nihayet herhangi bir bağlantı olmadığı ve bu bakılan sayılardan hiçbiri 34 olmadığı için bu sayının dizide olmadığı neticesine varılacaktır.

Dolayısıyla arama algoritmasını aşağıdaki şekilde yazabiliriz:

1. Sayının özetleme fonksiyonu (hashing function) değerini hesapla ve sayıyı bu bulduğun sırada ara
2. Sayı burada ise sayıyı buldun. Aramayı bitir.
3. Sayı yoksa ve bağlantı varsa bağlantıyı takip edip 2. Adıma git.
4. Bağlantı yoksa sayı dizide yok demektir.

LICH yöntemi yine sınıflandırma bakımından sabit metot (static method) olarak kabul edilebilir. Bundan kasıt, bir sayı bir sıraya yerleştikten sonra buradan hareket edemez, ilk defa yerleştiği yerde kalır.

LICH algoritmasının farklı uygulamalarına göre çeşitli isimlendirmelerin kullanılması da mümkündür. Örneğin, BLISCH algoritmasındaki B harfi, ingilizce bidirectional (iki yönlü) kelimesinden gelmektedir ve hashte olan taşmaların (overflow) iki yönde de olabileceğini (yani klasik lisch algoritmasındaki gibi her zaman sona değil de hashin başına da yönlendirilebileceğini) ifade eder.
Örneğin RLISCH çeşidinde ise R harfi Random (rastgele) kelimesinden gelmektedir ve ekleme yapılacak yer rast gele olarak seçilir.

SORU-8: EISCH (Early Insertion Standart Coalesced Hashing) hakkında bilgi veriniz.

Türkçeye, erken ekleme standart birleştirme özetlemesi olarak çevrilebilir. Bilgisayar bilimlerinde, özellikle dosya yönetimi konusunun (file organization) kullandığı bir özetleme (hashing) çakışması (collision) çözüm algoritmasıdır. Basitçe bir özetleme fonksiyonu (hashing function) sonucunda, çalışma olması durumunda (collision), dizinin sonundan başa doğru boş bulunan ilk yere yerleştirmeyi söyler.

Bu durumu bir örnek üzerinden anlamaya çalışalım.

Eklenecek olan sayılarımız 27, 18, 29, 28, 39, 13, 16, 42, 17 olsun. Kullanacağımız özetleme fonksiyonu ise H : K mod 11 olarak verilsin.

Bu duruma mod 11 için 11 farklı alandan oluşan boş bir dizimiz bulunacaktır:

Sıra	Anahtar	Bağlantı
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

Yukarıdaki boş tabloda, sıra sütunu, verilerin ekleneceği yeri, anahtar sütunu örnekte verilen sayıların nereye eklendiğini, bağlantı sütunu ise, arama ve ekleme işlemleri sırasında bir sayının beklenen yerde bulunamadığında nerede aranacağı bilgisini tutmaktadır.

Ayrıca yukarıda görülmeyen bir gösterici (Biz buna R diyelim), tablonun boş olan en alt elemanını göstermektedir. Yani şu anda R = 10 olarak başlayabiliriz.

Bu anlamda eisch algoritması bağlantılı çakışma çözümü (collision with links) yapmaktadır. Yani bir çakışma durumunda, konulması gereken sıraya değil farklı bir sıraya, bir sayı konulduğunda, bu yer bağlantı bilgisinde durmaktadır.

Yukarıdaki örnekte bulunan sayıları sırasıyla ekleyelim. İlk sayımız 27 için sıra hesaplanır ve 27 mod 11 = 5 olarak bulunur ve çakışma olmadan sorunsuz bir şekilde eklenir:

Sıra	Anahtar	Bağlantı
0		
1		
2		
3		
4		
5	27	
6		
7		
8		
9		
10		

Ardından eklenen sayımız 18 için sıra hesaplanır ve 18 mod 11 = 7 olarak bulunur ve çakışma olmadan sorunsuz bir şekilde eklenir:

Sıra	Anahtar	Bağlantı

0		
1		
2		
3		
4		
5	27	
6		
7	18	
8		
9		
10		

Ardından eklenen sayımız 29 için sıra hesaplanır ve 29 mod 11 = 7 olarak bulunur. Yukarıdaki görüldüğü üzere 7. Sıra doludur ve dolayısıyla bir çakışma durumu söz konusudur. Çakışma durumunda R ile gösterilen adrese bu çakışan yeni sayı yerleştirilir. Bu durumda R ile gösterilen ve boş hücrelerin en alttaki olan 10. Sıraya yerleştirme yapıyoruz. Ayrıca normalde 7. Sıraya yerleşmesi gereken 29 sayısını, farklı bir yere yerleştirdiğimiz için 7. Sırada bir bağlantı kurup, 7. Sırada bulunamayan sayılar için 10. Sıraya bakmasını söylüyoruz:

Sıra	Anahtar	Bağlantı
0		
1		
2		
3		
4		
5	27	
6		
7	18	10
8		
9		
10	29	

Sıradaki sayımı 28 mod 11 = 6. Sıraya problemsiz bir şekilde yerleşir:

Sıra	Anahtar	Bağlantı
0		
1		
2		
3		
4		
5	27	
6	28	

7	18	10
8		
9		
10	29	

Bir sonraki sayı olan 39 mod 11 = 6, yerleştirilirken çakışma olur ve şu anda R değeri olan, boş hücrelerin en altındaki 9. Sıraya yerleştirilir. Ayrıca 6. Sıraya yerleşmesi gerekirken, 9. Sıray a yerleşen bu sayı için 6′dan 9′a bir bağlantı kurulur:

Sıra	Anahtar	Bağlantı
0		
1		
2		
3		
4		
5	27	
6	28	9
7	18	10
8		
9	39	
10	29	

13 sayısı sorunsuz bir şekilde 13 mod 11 = 2. Sıraya yerleştirilir:

Sıra	Anahtar	Bağlantı
0		
1		
2	13	
3		
4		
5	27	
6	28	9
7	18	10
8		
9	39	
10	29	

16 sayısı ise 16mod11= 5. Sıra dolu olduğu için çakışır. Bu durumda R değeri olan en alttaki boş hücreye yani 8. Sıraya 16 sayısı yerleştirilip 5. Sıradan 8. Sıraya bağlantı kurulur:

Sıra	Anahtar	Bağlantı
0		
1		
2	13	
3		

4		
5	27	8
6	28	9
7	18	10
8	16	
9	39	
10	29	

42 sayısı da benzer şekilde problemli bir sayıdır. 42 mod 11 = 9. Sıra doludur ve şu anda en alttaki boş hücre 4′tür. Bu durumda sayı 4. Sıraya yerleştirilerek 9. Sıradan buraya bir bağlantı oluşturulur:

Sıra	Anahtar	Bağlantı
0		
1		
2	13	
3		
4	42	
5	27	8
6	28	9
7	18	10
8	16	
9	39	4
10	29	

Son olarak gelen 17 sayısı, yerleşmesi gereken 17 mod 11 = 6. Sıra dolu olduğu için en alttaki boş hücre olan 3. Sıraya yerleştirilir. Ardından 6. Sıradan buraya bağlantı oluşturulması gerekir. Ancak yukarıda görüldüğü üzere 6. Sırada zaten bir bağlantı vardır ve bu bağlantı 9. Sıraya işaret etmektedir. Bu durumda çözüm olarak yeni yerleştirilen sıra ile 6. Sıradaki bağlantı değiştirilir ve 6. Sırada zaten var olan bağlantı yeni yerleştirilen sayıya konulur:

Sıra	Anahtar	Bağlantı
0		
1		
2	13	
3	17	9
4	42	
5	27	8
6	28	3
7	18	10
8	16	
9	39	4
10	29	

Yukarıdaki bu problemli durumda görüldüğü üzere 17 sayısı 3. Sıraya yerleştirilmiş dolayısıyla 6′da bulunan eski bağlantı değeri olan 9, 3. Sıradaki bağlantı olarak taşınmış bunun yerine de yeni sayıyı yerleştirdiğimiz sıra olan 3 yazılmıştır.

Yukarıdaki bu durumu, bağlantı seviyesinin 2′den fazla olmaması olarak da yorumlamak mümkündür. Yani eski durumda şayet 28 → 39 → 42 → 17 bağlantısı kurulacak olsaydı, bu durumda 3 atlama yapılması gerekecekti. Bunu engellemek için yukarıdaki çözüm kullanılmıştır.

Yukarıdaki örnekte sayıların nasıl yerleştirildiğini gördük. Bu yöntemi özetleyecek olursak:

1. Yerleşecek sayının (anahtarın) özetleme fonksiyonu değerini (hashing function) hesapla ve buraya yerleştir.
2. Şayet burası doluysa, dizideki boş en alt hücreye yerleştir ve bu yerleştirdiğin yere, normalde yerleşmesi gereken yerden bir bağlantı kur
3. Sayının, normalde yerleşmesi gereken yerde bir bağlantı varsa, bu bağlantıyı yeni yerleşen sayının yanına taşı.

Yukarıdaki bu 3 basit adım takip edilerek EISCH algoritmasına göre yerleştirme işlemi yapılabilir.

Yerleşen bu sayıların aranması ise oldukça basittir.

Örneğin aranan sayının 17 olduğunu düşünelim. 17 mod 11 = 6 olarak bulunur ve 6. Sırada bu sayı aranır. Sayı burada olmadığı için buradaki bağlantı olan 3. Sıraya gidilip bakılır ve 17 sayısı bulunmuş olunur.

Benzer şekilde, dizide hiç bulunmayan bir sayı aranırsa, bağlantısı olmayan bir sıraya gelindiğinde bu sayının dizide olmadığına kanaat getirip durulabilir. Örneğin dizimizde olmayan 34 sayısını aramak istediğimizi düşünelim.

34 mod 11 = 3 bulunup bu sıraya bakılacak, bu sırada 34 olmadığı görülünce bağlantının gösterdiği 9. Sıraya oradan da 4. Sıraya bakılacak ve nihayet herhangi bir bağlantı olmadığı ve bu bakılan sayılardan hiçbiri 34 olmadığı için bu sayının dizide olmadığı neticesine varılacaktır.

Dolayısıyla arama algoritmasını aşağıdaki şekilde yazabiliriz:

1. Sayının özetleme fonksiyonu (hashing function) değerini hesapla ve sayıyı bu bulduğun sırada ara
2. Sayı burada ise sayıyı buldun. Aramayı bitir.
3. Sayı yoksa ve bağlantı varsa bağlantıyı takip edip 2. Adıma git.
4. Bağlantı yoksa sayı dizide yok demektir.

Yukarıda anlatılan EISCH yöntemine göre aranan sayıların dizide, alttan yukarı doğru doldurulması gerekmektedir. Ayrıca EISCH yönteminin bir dezavantajı çakışan ve çakışmayan bütün sayıların karışık bir şekilde iç içe durmasıdır. Bu problem LICH yönteminde veya CCI yönteminde çözülmüştür.

EISCH yöntemi yine sınıflandırma bakımından sabit metot (static method) olarak kabul edilebilir. Bundan kasıt, bir sayı bir sıraya yerleştikten sonra buradan hareket edemez, ilk defa yerleştiği yerde kalır.

SORU-9: Çift Özetleme (Double Hashing) hakkında bilgi veriniz.

Bilgisayar bilimlerinde kullanılan özetleme fonksiyonları, genellikle büyük bir verinin daha küçük bir hale getirilmesine yarar. Bu anlamda özetleme fonksiyonları veri doğrulama (data verification) , veri bütünlüğü (data integrity), veri güvenliği (security) ve şifreleme (encryption) gibi pek çok alanda kullanılırlar.

Özetleme fonksiyonlarının bir problemi, büyük bir veriyi özetledikten sonra, çakışma olması durumudur. Çalışma (collision) kısaca aynı özet değerine sahip iki farklı verinin olmasıdır.

Örneğin en basit özetleme fonksiyonlarından birisi olan kalan (mod) işlemini ele alalım. 0 ile 100 arasındaki sayıları, 0 ile 10 arasındaki sayılarla özetlemek istersek, mod 10 kullanmamız mümkündür. Bu durumda her sayının 0 – 10 aralığında bir karşılığı bulunacaktır.

Ancak 41 mod 10 ile 51 mod 10 aynı sonucu verir. Bu durumda bir çakışma olmuş denilebilir.

Çakışmayı engellemek için veri yapılar üzerinde sondalama yöntemleri kullanılabilir. En meşhurları olan doğrusal sondalama (linear probing) ve ikinci dereceden sondalama (quadratic probign) yöntemleri bu problemin çözümü için geliştirilmiş yöntemlerdir.

Bu yazının konusu olan çift özetleme (double hashing) yöntemi de işte tam bu noktada devreye girer. Yani bir şekilde özetleme fonksiyonundan çıkan sonuçların, çakışması (collision) durumunda, ikinci ve farklı bir özetleme fonksiyonu kullanılarak veri yapısı üzerinde farklı bir noktada arama veya veri ekleme işlemine devam edilebilir.

Örneğin veri yapısına ekleme işlemini ele alalım. Verinin hangi adrese ekleneceğini bulmak için öncelikle anahtar (key) bir özetleme fonksiyonuna sokulur. Bu ilk özetleme fonksiyonuna Ö1 ismini verelim.

Adres = Ö1 (anahtar) formülü ile adresi buluruz. Diyelim ki bu adres dolu ve buraya yeni verimizi ekleyemiyoruz. Bu durumda ikinci bir adres aranmalıdır. İşte bu noktada ikinci özetleme fonksiyonu $Ö_2$ devreye girer ve verinin yerleştirilebileceği boş bir adres bulunana kadar bu fonksiyon kullanılmaya devam edilir.

Bu durumu 10 adet hücresi bulunan boş bir veri yapısı üzerinden sırasıyla 21,31,41,51 sayılarının eklenmesi şeklinde görelim. Ö1 fonksiyonu olarak

Adres = anahtar mod 10

Ve ikinci özetleme fonksiyonu olarak Ö2 :

Adres = ((anahatar mod 7) x 3) mod 10

Fonksiyonlarını alalım. Bu fonksiyonlar örnek olarak alınmıştır ve farklı fonksiyonlar üzerinden de çift özetleme (double hashing) yapılabilir.

Sırasıyla sayılarımızın üzerinden geçiyoruz. İlk sayımız 21 mo 10 = 1 olarak bulunur.

0	
1	21
2	
3	
4	
5	
6	
7	
8	
9	

İlk özetleme fonksiyonu sonucu olarak 1 numaralı adrese yerleştirilir. Ardından ikinci sayıya geçilir:

31 mod 10 = 1 bulunur ve bu adres dolu olduğu için çakışma olur. Çözüm olarak ikinci özetleme fonksiyonu kullanılır.

((31 mod 7) x 3) mod 10 = 9 olarak bulunur ve bu adrese yerleştirilir:

0	
1	21
2	
3	
4	
5	
6	
7	
8	
9	31

Ardından 41 sayısı için 1. özetleme fonksiyonu çalıştırılır ve 1 numaralı adres dolu olduğu için çakışma oluşur. Çözüm olarak ikinci özetleme fonksiyonu çalıştırılır:

((41 mod 7) x 3) mod 10 = 8

0	
1	21
2	
3	
4	
5	
6	
7	
8	41

9	31

Benzer şekilde 51 için çakışma olur ve ikinci özet fonksiyonu çalıştırılır.

((51 mod 7) x 3) mod 10 = 6

0	
1	21
2	
3	
4	
5	
6	51
7	
8	41
9	31

Görüldüğü üzere ikinci özetleme fonksiyonu, ilkinde bir çakışma olması halinde kullanılmaktadır. Peki acaba ikinci özetleme fonksiyonunda da çakışma olursa ne yapılır?

Bu durumu örneğimize devam edip 61 sayısını eklemek istediğimizde görebiliriz.

((61 mod 7) x 3) mod 10 = 8

Bulunacaktır ve 8 numaralı adres doludur. Bu durumda ikinci özetleme fonksiyonuna ikinci kere sokularak farklı bir adres aranır:

((8 mod 7) x 3) mod 10 = 3 olarak bulunur ve bu adrese yerleştirilir.

0	
1	21
2	
3	61
4	
5	
6	51
7	
8	41
9	31

Görüldüğü üzere ikinci özetleme fonksiyonunun ilk özetleme fonksiyonu ile aralarında asal olması, belirli bir adres dizilimine girilmesine engellemekte bu sayede veri yapısı üzerinde boş yer bulunuyorsa mutlaka belirli bir denemeden sonra bu adrese erişilmesi garanti edilmiş olmaktadır.

SORU-10: İkinci Dereceden Sondalama (Quadratic Probing) hakkında bilgi veriniz.

Özellikle özetleme fonksiyonlarının (hashing functions) bilgileri sınıflandırması sırasında kullanılan formülün ikinci dereceden olması durumudur.

Özetleme fonksiyonlarında, sık kullanılan doğrusal sondalama (linear probing) yönteminin tersine, bir bilgiyi tasnif ederken, ardışık olarak veriler üzerinde hareket etmez, bunun yerine her defasında baktığı uzaklığı ikinci dereceden bir denklem ile arttırır.

Konuyu anlamaya öncelikle doğrusal fonksiyonları hatırlayarak başlayalım.

Bilindiği üzere doğrusal fonksiyonlar :

y = ax + b şeklinde yazılabilen birinci dereceden fonksiyonlardır. Bu durumda özetleme fonksiyonu veriyi sınıflandırırken bir çakışma (collision) olması durumunda bir sonraki veri hücresine bakar, ve bu şekilde aranan veriye ulaşana kadar devam eder. Örneğin 10 hücreli bir sınıflama için mod 10 fonksiyonunu kullanacağımızı düşünelim.

Bu durumda ilk başta aşağıdaki şekilde boş olan veri yapımıza sırasıyla 21,31,41,51 sayılarının geldiğini kabul edelim.

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Dikkat edileceği üzere bu sayılar özellikle çakışma olsun diye mod 10 fonksiyonuna konulduğunda hep 1 olarak çıkan sayılardan seçilmiştir.

Şimdi bu sayıları yerleştirecek olursak ilk gelen sayı 21 mod 10 = 1 olduğu için 1. Hücreye yerleştirilecektir.

0	
1	21
2	
3	
4	
5	
6	
7	
8	

9	

Ardından gelen 31 sayısı yine 1. Hücreye yerleştirilmek istenecek ancak burası dolu olduğu için doğrusal sondalama kullanarak bir sonraki hücreye yerleştirme yapılıyor:

0	
1	21
2	31
3	
4	
5	
6	
7	
8	
9	

Diğer sayılar için durum değişmiyor ve aynı yaklaşım izlenmeye devam ediliyor:

0	
1	21
2	31
3	41
4	51
5	
6	
7	
8	
9	

Görüldüğü üzere elimizdeki sayıların tamamı başarılı bir şekilde yerleştirilmiştir. Konuyu daha iyi anlayabilmek için farklı bir örnek olarak 33 sayısını yerleştirmek istediğimizi düşünelim. Bu durumda 33 mod 10 = 3 hücresi dolu olduğu için işlem değişmeden yukarıda olduğu gibi uygulanacaktır:

0	
1	21
2	31
3	41
4	51
5	33
6	
7	
8	
9	

İkinci dereceden sondalama (quadratic probing)

Doğrusal sondalamayı anladıktan (hatırladıktan) sonra ikinci dereceden sondalamadan bahsedebiliriz. Buradaki yaklaşımda kullanılan formül bir özetleme fonksiyonunda geçildikten sonra çakışma olması durumunda (collision) sürekli olarak bir sonraki hücreye bakmak yerine, her defasında üssel fonksiyon değeri kadar ilerlemektir.

Örneğin yukarıdaki sayılar için aynı veri yapısına ikinci dereceden sondalama ile ekleme işlemi yapalım:

0	
1	21
2	
3	
4	
5	
6	
7	
8	
9	

İlk gelen sayı, bir çakışma olmadığı için doğal hücresine yerleştiriliyor. Ardından gelen 31 sayısı için çakışma oluyor. Bu durumda sonraki bakılacak olan hücrenin karesi alınıyor, şu anda ilk çakışma yaşandığı için $1^2 = 1$ yani doğal hücreden bir sonraki hücreye bakıyoruz:

0	
1	21
2	31
3	
4	
5	
6	
7	
8	
9	

Bu hücre boş olduğu için yerleştirme işlemi tamamlanıyor ve sıradaki sayıya geçiliyor.

Sayımız 41 ve doğal hücresi olan 1. Adreste çakışma yaşanıyor,

ardından $1^2 = 1$ sonraki hücrede de (yani 2. Hücre) çakışma yaşanıyor,

ardından gelen $2^2 = 4$ sonraki hücreye bakılıyor. Dolayısıyla doğrusal sondalamada olduğu gibi 3. Hücreye bakmak yerine 5. Hücreye bakıyoruz:

0	
1	21
2	31
3	

4	
5	41
6	
7	
8	
9	

Sırada 51 bulunuyor ve benzer şekilde doğal hücresi dolu, $1^2 = 1$ sonraki hücre dolu, $2^2 = 4$ sonraki hücre dolu ve nihayet $3^2 = 9$ sonraki hücre boş bulunup yerleştiriliyor.

0	51
1	21
2	31
3	
4	
5	41
6	
7	
8	
9	

Yukarıdaki yerleştirme işlemi sırasında mod10 kullanıldığı unutulmamalıdır, bu yüzden 0. Hücreye yerleştirme işlemi yapılmıştır.

Doğrusal sondalama örneğinde olduğu gibi, 33 sayısını da eklemek istediğimizde doğal hücresi boş olduğu için herhangi bir sondalamaya gerek kalmadan yerleştirme işlemi yapılabilir:

0	51
1	21
2	31
3	33
4	
5	41
6	
7	
8	
9	

Doğrusal sondalamada özetleme fonksiyonundan sonra sırasıyla veri yapısındaki hücrelere bakılır. Bu durum için aşağıdakine benzer bir döngü yazılması gerekir:

```
adres = anahtar % 10;
while(dolu(adres++)){
    adres %= 10;
}
ekle(anahtar,adres);
```

Görüldüğü üzere, öncelikle özetleme fonksiyonuna (h) anahtar verilip doğal adres bulunuyor. Bu adresin dolu olması ve sonraki adreslerin de dolu olması durumunda adres değeri arttırılarak devam ediliyor.

İkinci dereceden sondalama kullanılsaydı, bu kodu aşağıdaki şekilde yazmamız gerekecekti:

```
adres = anahtar % 10;
int artis = 0;
while(dolu(adres)){
    adres = (adres + artis * artis) % 10;
    artis ++;
}
ekle(anahtar,adres);
```

Yukarıdaki yeni haliyle kodumuz, her defasında artış miktarının karesi kadar sonraki hücreye bakmaktadır.

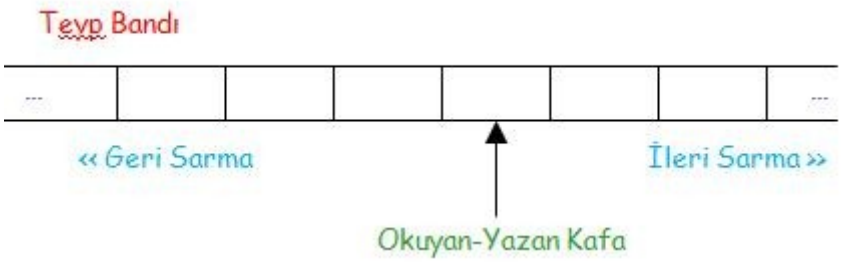
Yukarıdaki müsvedde kodlar (pseudo codes) ve örnekler fikir vermesi açısından yazılmış olup, özetleme fonksiyonu, doğrusal sondalama ve ikinci dereceden sondalama işlemleri için farklı fonksiyonlar kullanılabilir. Buradaki amaç, ikinci dereceden sondalamada kullanılan fonksiyonun ikinci derece bir denklem olmasıdır.

SORU-11: Turing Makinesi (Turing Machine) hakkında bilgi veriniz.

Bilgisayar bilimlerinin önemli bir kısmını oluşturan otomatlar (Automata) ve Algoritma Analizi (Algorithm analysis) çalıştırmalarının altındaki dil bilimin en temel taşlarından birisidir.1936 yılında Alan Turing tarafından ortaya atılan makine tasarımı günümüzde pekçok teori ve standardın belirlenmesinde önemli rol oynar.

Turing Makinesinin Tanımı

Basitçe bir kafadan (head) ve bir de teyp bandından (tape) oluşan bir makinedir.



Makinede yapılabilecek işlemler

- Yazmak
- Okumak
- Bandı ileri sarmak
- Bandı geri sarmak

şeklinde sıralanabilir.

Chomsky hiyerarşisi ve Turing Makinesi

Bütün teori bu basit dört işlem üzerine kurulmuştur ve sadece yukarıdaki bu işlemleri kullanarak bir işin yapılıp yapılamayacağı veya bir dilin bu basit 4 işleme indirgenip indirgenemeyeceğine göre diller ve işlemler tasnif edilmiştir.

Bu sınıflandırma yukarıdaki venn şeması ile gösterilmiştir. Aynı zamanda chomsky hiyerarşisi (chomsky hierarchy) için 1. seviye (type-1) olan ve Turing makinesi ile kabul edilebilen diller bütün tip-2 ve tip-3 dilleri yani içerk bağımsız dilleri ve düzenli dilleri kapsamaktadır. Ayrıca ilave olarak içerik bağımsız dillerin işleyemediği (üretemediği veya parçalayamadığı (parse)) anbncn şeklindeki kelimeleri de işleyebilmektedir. Düzenli ifadelerin işleyememesi konusunda bilgi için düzenli ifadelerde pompalama savı (pumping lemma in regular expressions) ve içerik bağımsız dillerin işlemeyemesi için de içerik bağımsız dillerde pompalama savı (pumping lemma for CFG) başlıklı yazıları okuyabilirsiniz.

Turing Makinesinin Akademik Tanımı

Turing makineleri literatürde akademik olarak aşağıdaki şekilde tanımlanır:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \Diamond, F)$$

Burada M ile gösterilen makinenin parçaları aşağıda listelenmiştir:

Q sembolü sonlu sayıdaki durumların kümesidir. Yani makinenin işleme sırasında aldığı durumardır.

Γ sembolü dilde bulunan bütün harfleri içeren alfabeyi gösterir. Örneğin ikilik tabandaki sayılar ile işlem yapılıyorsa {0,1} şeklinde kabul edilir.

Σ sembolü ile makineye verilecek girdiler (input) kümesi gösterilir. Girdi kümesi dildeki harfler dışında bir sembol taşıyamayacağı için $\Sigma \subseteq \Gamma$ demek doğru olur.

δ sembolü dilde bulunan ve makinenin çalışması sırasında kullanacağı geçişleri (transitions) tutmaktadır.

◊ sembolü teyp bandı üzerindeki boşlukları ifade etmektedir. Yani teyp üzerinde hiçbir bilgi yokken bu sembol okunur.

q_0 sembolü makinenin başlangıç durumunu (state) tutmaktadır ve dolayısıyla $q_0 \subseteq Q$ olmak zorundadır.

F sembolü makinenin bitiş durumunu (state) tutmaktadır ve yine $F \subseteq Q$ olmak zorundadır.

Örnek Turing Makinesi

Yukarıdaki sembolleri kullanarak örnek bir Turing makinesini aşağıdaki şekilde inşa edebiliriz.

Örneğin basit bir kelime olan a* düzenli ifadesini (regular expression) Turing makinesi ile gösterelim ve bize verilen aaa şeklindeki 3 a yı makinemizin kabul edip etmediğine bakalım.

Tanım itibariyle makinemizi aşağıdaki şekilde tanımlayalım:

$M = \{ \{q_0,q_1\} , \{ a \} , \{ a,x \} , \{ q_0\ a \rightarrow a\ R\ q_0 , q_0\ x \rightarrow x\ L\ q_1\} , q_0 , x , q_1 \}$

Yukarıdaki bu makineyi yorumlayacak olursak:

Q değeri olarak $\{q_0,q_1\}$ verilmiştir. Yani makinemizin ik idurumu olacaktır.

Γ değeri olarak { a,x } verilmiştir. Yani makinemizdeki kullanılan semboller a ve x'ten ibarettir.

Σ değeri olara {a} verilmiştir. Yani makinemize sadece a girdisi kabul edilmektedir.

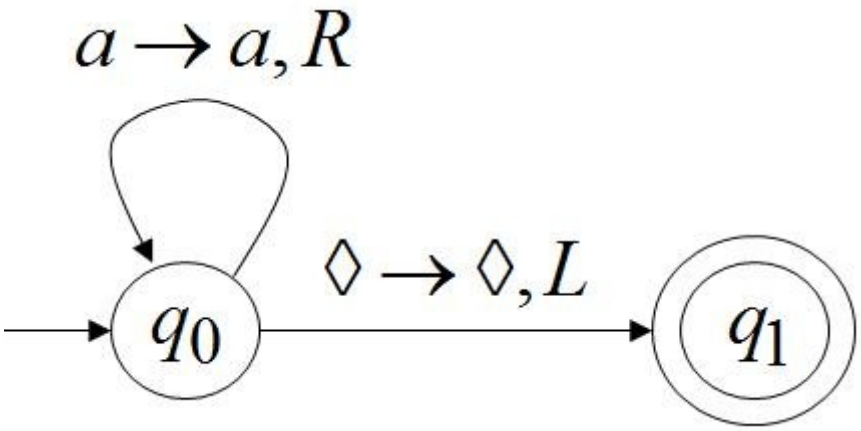
δ değeri olarak iki geçiş verilmiştir $\{ q_0\ a \rightarrow a\ R\ q_0 , q_0\ x \rightarrow x\ L\ q_1\}$ buraadki R sağa sarma L ise sola sarmadır ve görüleceği üzere Q değerindeki durumlar arasındaki geçişleri tutmaktadır.

◊ değeri olarak x sembolü verilmiştir. Buradan x sembolünün aslında boş sembolü olduğu ve bantta hiçbir değer yokken okunan değer olduğu anlaşılmaktadır.

q_0 ile makinenin başlangıç durumundaki hali belirtilmiştir.

F değeri olarak q_1 değeri verilmiştir. Demek ki makinemiz q_1 durumuna geldiğinde bitmektedir (halt) ve bu duruma gelmesi halinde bu duruma kadar olan girdileri kabul etmiş olur.

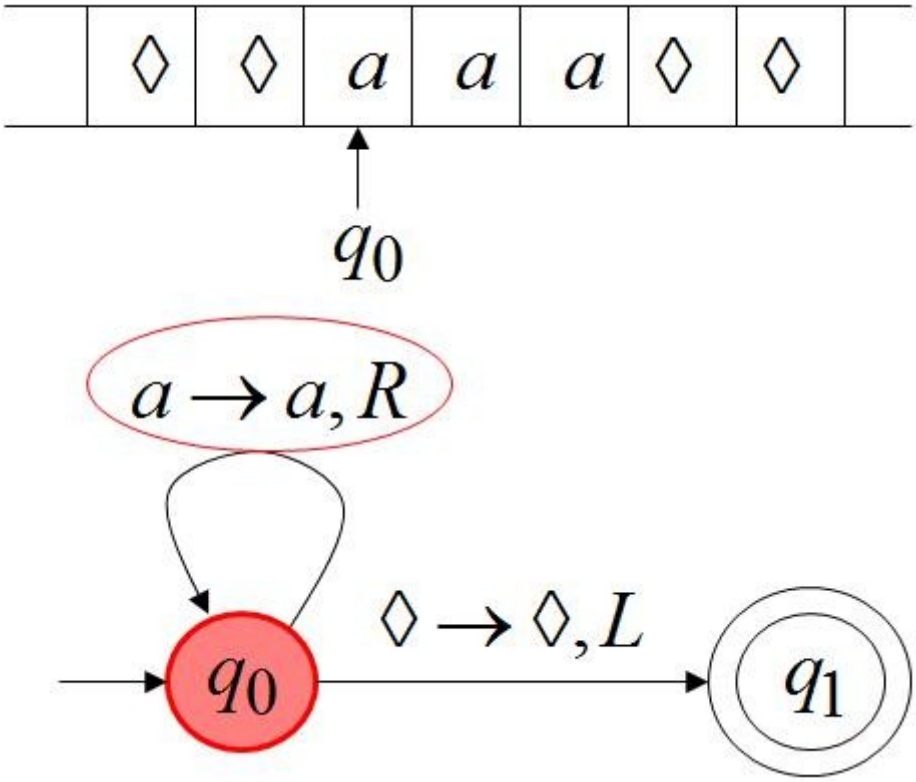
Yukarıdaki bu tanımı görsel olarak göstermek de mümkündür:



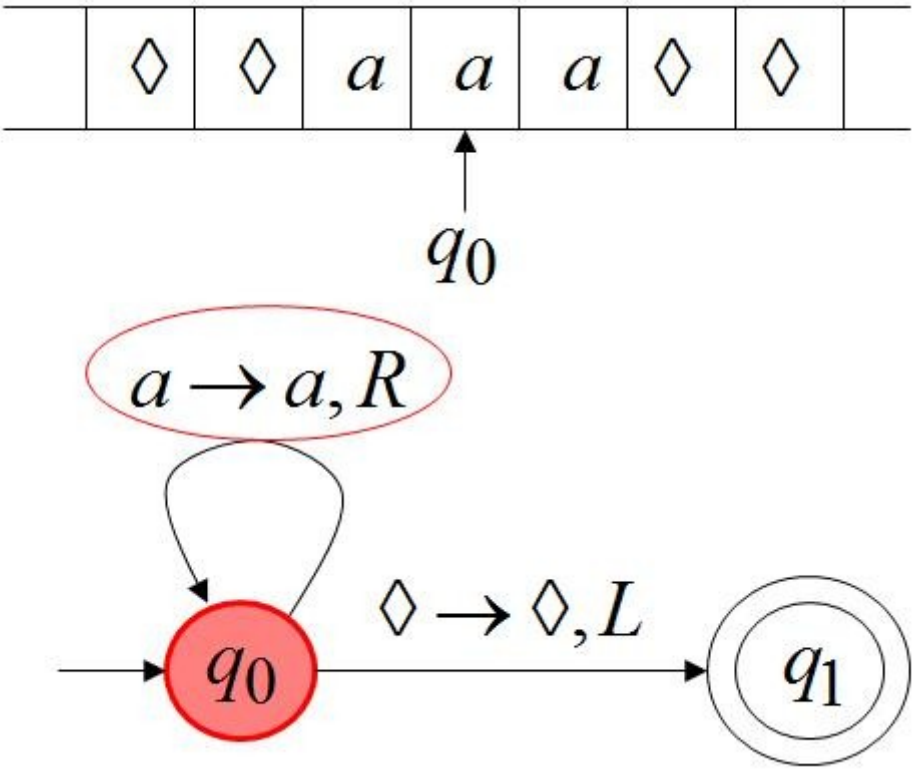
Yukarıdaki bu temsili resimde verilen turing makinesi çizilmiştir.

Makinemizin örnek çalışmasını ve bant durumunu adım adım inceleyelim.

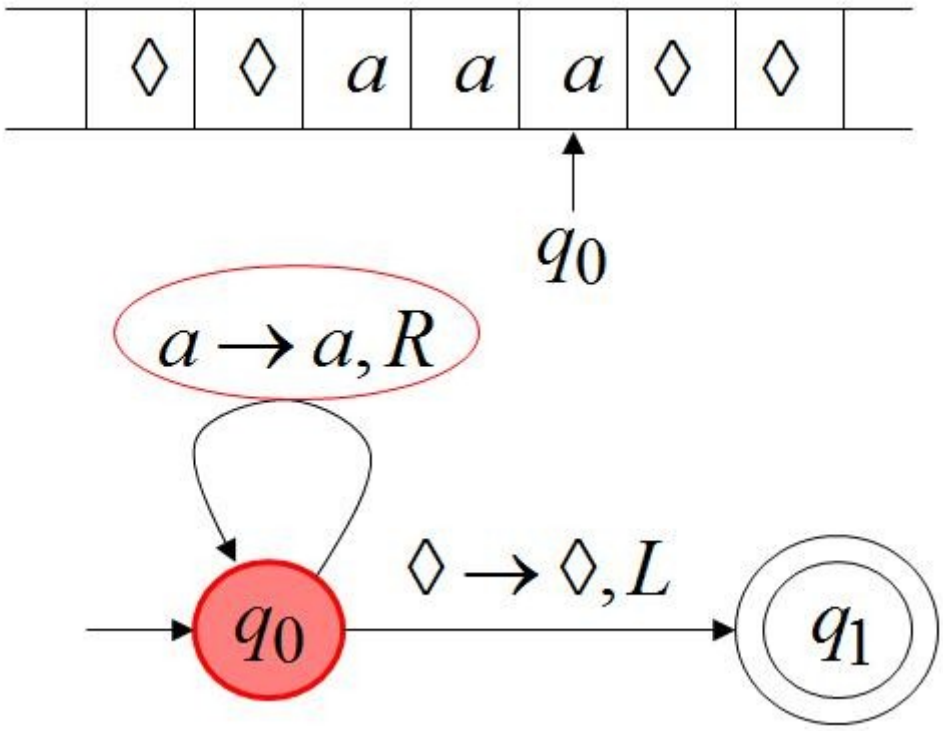
Birinci adımda bandımızda aaa (3 adet a) yazılı olduğunu kabul edelim ve makinemizin bu aaa değerini kabul edip etmeyeceğini adım adım görelim. Zaten istediğimiz de aaa değerini kabul eden bir makine yapabilmekti.



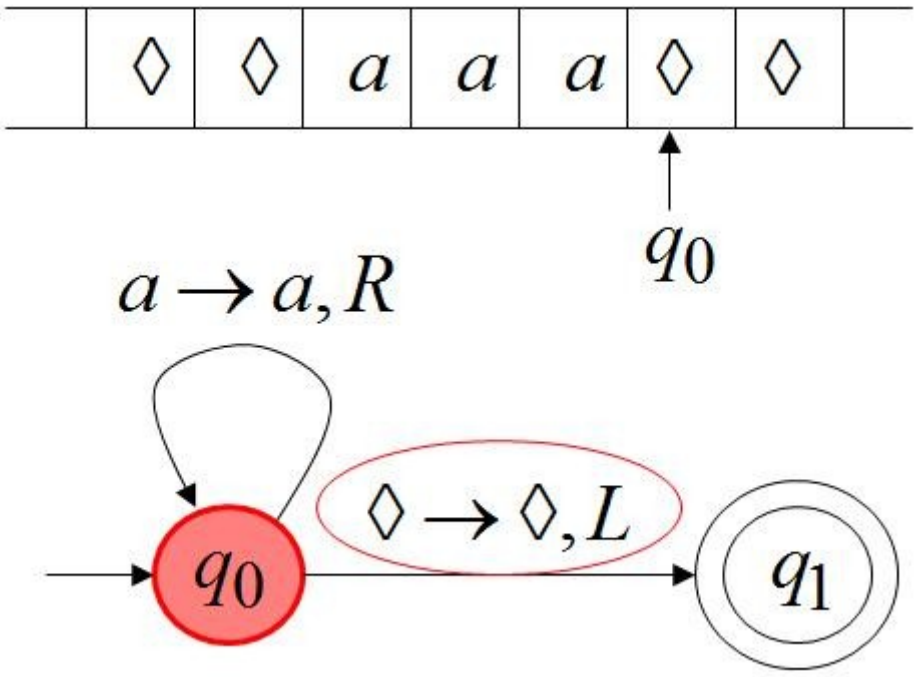
Yukarıdaki ilk durumda bant üzerinde beklenen ve kabul edilip edilmeyeceği merak edilen değerimiz bulunuyor. Makinemizin kafasının okuduğu değer a sembolü. Makinemizin geçiş tasarımına göre q_0 halinde başlıyoruz ve a geldiğinde teybi sağa sarıp yine q_0 durumunda kalmamız gerekiyor.



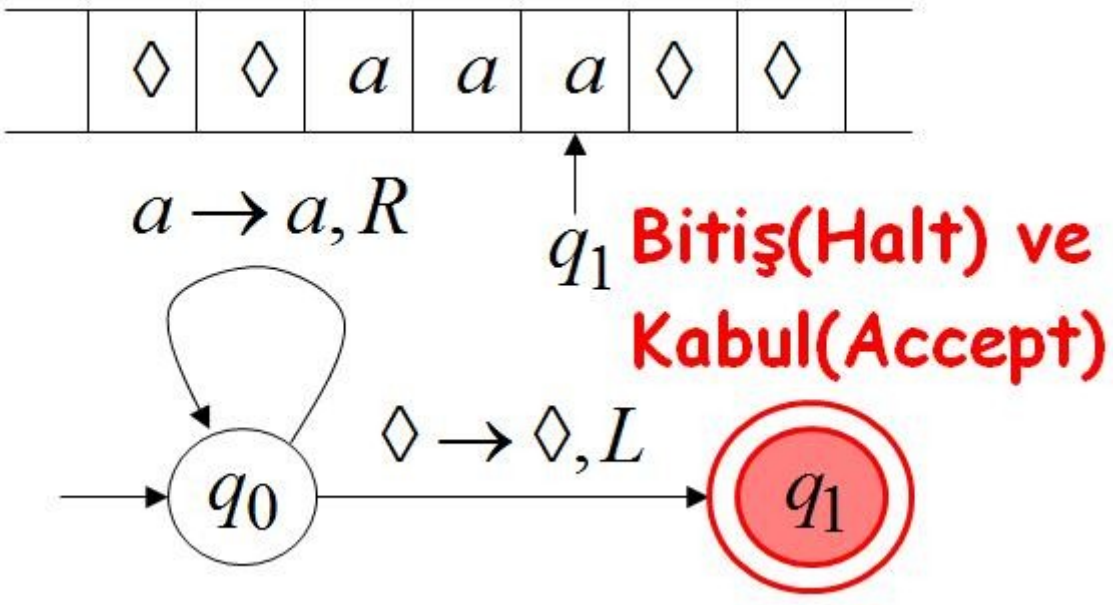
Yeni durumda kafamızın okuduğu değer banttaki 2. a harfi ve bu durumda yine q_0 durumundayken teybi sağa sarıp yine q_0 durumunda kalmamız tasarlanmış



3. durumda kafamızın okuduğu değer yine a sembolü olmakta ve daha önceki 2 duruma benzer şekilde q_0 durumundayken a sembolü okumanın sonucu olarak teybi sağa sarıp q_0 durumunda sabit kalıyoruz.



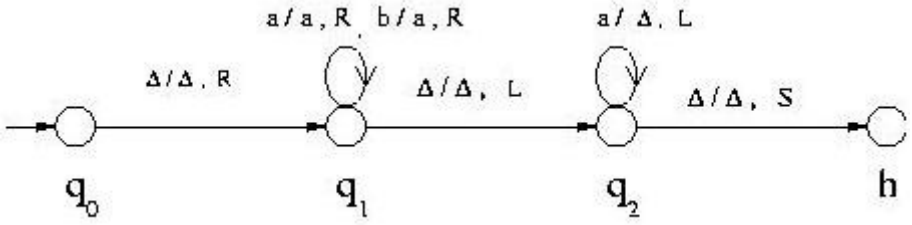
4. adımda teypten okuduğumuz değer boşluk sembolü x oluyor. Bu değer makinemizin tasarımında q_1 durumuna gitmemiz olarak tasarlanmış ve teybe sola sarma emri veriyoruz.



Makinenin son durumunda q_1 durumu makinenin kabul ve bitiş durumu olarak tasarlanmıştı (makinenin tasarımındaki F kümesi) dolayısıyla çalışmamız burada sonlanmış ve giriş olarak aaa girdisini kabul etmiş oluyoruz.

2. Örnek

Hasan Bey'in sorusu üzerine bir örnek makine daha ekleme ihtiyacı zuhur etti. Makinemiz {a,b} sembolleri için çalışsın ve ilk durum olarak bandın en solunda başlayarak bantta bulunan sembolleri silmek için tasarlansın. Bu tasarımı aşağıdaki temsili resimde görülen otomat ile yapabiliriz:



Görüldüğü üzere makinemizde 4 durum bulunuyor, bunlardan en sağda olan h durumu bitişi (halt) temsil ediyor. Şimdi bu makinenin bir misal olarak "aabb" yazılı bir bantta silme işlemini nasıl yaptığını adım adım izah etmeye çalışalım.

Aşağıda, makinenin her adımda nasıl davranacağı bant üzerinde gösterilmiş ve altında açıklanmıştır. Sarı renge boyalı olan kutular, kafanın o anda üzerinde durduğu bant konumunu temsil etmektedir.

Netice olarak Hasan Bey'in sorusuna temel teşkil eden ve örneğin q1 üzerindeki döngülerden birisi olan b/a,R geçişi, banttan b okunduğunda banta a değerini yaz manasındadır.

Yine bu yazıya yapılan yorumlarda sorulan bir sorunun cevabını aşağıdaki bağlantıda çözdüm. Soru, eşit a ve b içeren Turing makinesinin tasarımı idi, bağlantıya tıklayarak okuyabilirsiniz.

SORU-12: fstream (File Stream, Dosya Akışı) hakkında bilgi veriniz.

Özellikle C++ dilinde dosyalara erişmek ve dosyalar üzerinde işlem yapmak için çeşitli fonksiyonlardan oluşan bir kütüphanenin ismidir.

Aslında bilgisayarlardaki giriş çıkış işlemlerini (I/O input/output) dört ana başlıkta toplamak mümkündür.

- Standart giriş çıkış işlemleri (klavye ve ekran) (Standard input output)

- Dosya giriş çıkış işlemleri (İşletim sisteminin dosya yapısındaki bir dosyadan okumak ve bu dosyaya yazmak şeklinde) (File input output)
- Hafıza giriş çıkış işlemleri (Hafızanın herhangi bir bölgesinden veya bazı durumlarda farklı işlemlerin (process) aralarında iletişimi için hafızadan okumak ve yazmak şeklindeki giriş çıkışlar (Memory input output)
- Ağ giriş çıkış işlemleri (Ağda bulunan başka bir bilgisyara belirlenmiş bir protokol çerçevesinde (Örneğin TCP/IP) ile veri göndermek ve almak şeklindeki giriş çıkış işlemleri) (Network input output)

Bu yazının amacı, dosyala giriş çıkış işlemlerini içeren fstream kütüphanesini tanıtmaktır.

Yine başlamadan önce bilinmesi gerekir ki dosyalar programlama dillerinde iki ana grup altında incelenebilir:

- Metin dosyaları (Text Files)
- İkili dosyalar (Binary Files)

Metin dosyalarını basitçe bir metin editörüyle açıp okuduğumuz (örneğin notepad, vi gibi) dosyalar olarak düşünebiliriz. Bu dosyalarda veri ardışık bir şekilde (sequential) dosyaya yazılmıştır. Dosyadaki herhangi bir veriye erişmek için dosyanın başından o veriye kadar ilerlemek gerekir.

İkili dosyalarda ise, dosyada bulunan verilere erişmeyi rastgele olarak yapabiliriz. Örneğin dosyanın 100. byte'ındaki bir veriye tek seferde erişilebilir. Ancak ikili dosyaların okunması ve yazılması metin dosyaları kadar basit değildir. Bu dosya tiplerinin kodlanma durumlarına göre özel editörler ile açılması gerekir. Örneğin herhangi bir ofis uygulaması ile (openoffice, microsoft office gibi) kaydettiğiniz bir dökümanı basit bir metin editörü ile açmayı deneyebilirsiniz. Dosya açıldığında sizin için anlamsız semboller belirecektir. Sembollerin bu şekilde belirmesinin sebebi aslında dosyada bulunan değerlere ASCII tablosundan birer karşılık aranmasından kaynaklanmaktadır. Oysaki bir ikili dosyadaki değerlerin ASCII karşılığı olması gerekmez.

Dosyalama işlemlerini içeren fstream kütüphanesini bu yazı kapsamında 4 grupta inceleyeceğiz.

1. Dosyaların açılması
2. Dosyaların kapatılması
3. Dosyalardan okuma ve yazma işlemleri
4. İlave fonksiyonlar

Dosyaların açılması

Dosyalar 3 farklı şekilde açılabilir:

1. Okuma şeklinde (Reading Mode)
2. Yazma şeklinde (Writing Mode)
3. Ekleme şeklinde (Append Mode)

Basitçe okuma şeklinde açılan bir dosyadan sadece veri okunabilir. Yazma şeklindeki dosyalara veri yazılabilir ve dosyada daha önceden bulunan bütün veriler silinir (bir anlamda

üzerine yazılmış olur). Ekleme şeklinde açılan dosyalarda ise mevcut veri saklanır ve yazılan veriler dosyanın sonuna ilave edilir.

Dosyaların iki farklı tutulma şekli bulunduğunu daha önce görmüştük. Aşağıda dosyaların bu tutulma şekillerine (ikili veya metin) ve yukarıdaki açılma şekillerine göre fstream kütüphanesinden hangi fonksiyonla açıldığını görmekteyiz:

	Metin Dosyaları	**İkili Dosyalar**
Yazma Şekli (Write mode)	Yazmak için ofstream out (“dosya.txt”); veya ofstream out; out.open(“dosya.txt”);	Yazmak için ofstream out (“dosya.txt”,ios::binary); veya ofstream out; out.open(“dosya.txt”, ios::binary);
Ekleme Şekli (Append Mode)	ofstream out(“dosya.txt”,ios::app); veya ofstream out; out.open(“dosya.txt”, ios::app);	ofstream out (“dosya.txt”,ios::app| ios::binary); veya ofstream out; out.open(“dosya.txt”, ios::app | ios::binary);
Okuma Şekli (Read Mode)	ifstream in (“dosya.txt”); veya ifstream in ; in.open(“dosya.txt”);	ifstream in (“dosya.txt”, ios::binary); veya ifstream in ; in.open(“dosya.txt”, ios::binary);

Yukarıdaki tablodaki kod örnekleri kullanılarak bir dosya okuma, yazma veya ekleme şekillerinde açılabilir.

Dosyaların Kapatılması

Dosya işlemleri sırasında özellikle bir dosyaya veri yazıldıktan sonra dosyanın kapatılması çok önemlidir. Çünkü işletim sistemi bir dosyaya yazılacak olan verileri doğrudan yazmak yerine hafızada bekletebilir. Bunun sebebi dosyaya yazılacak veriler üzerinde ileride bir değişiklik olma ihtimali ve bu durumda dosyaya erişmeden hafıza üzerinden ilgili değişikliğin yapılmasıdır.

Ancak dosya kapatıldığı zaman, programımızda dosyaya yazılmasını istediğimiz şeylerin tamamının dosyaya yazılmış olduğundan emin olabiliriz. Aksi halde yukarıda bahsedilen durum gibi dosyaya yazılması beklenen herşey yazılmış olmayabilir.

fstream kütüphanesinde dosyaların kapatılması için close() fonksiyonu bulunur. Bir dosyanın hangi şekilde açıldığına bakılmaksızın (ekleme, yazma veya okuma) ve dosyanın tipine bakılmaksızın (metin veya ikili dosya) bu fonksiyon kullanılabilir.

in.close();

veya out.close();

şeklinde dosya kapatılabilir.

Dosyalardan okuma ve yazma işlemleri

Veri Tipi	Okuma fonksiyonları	Yazma fonksiyonları
char	get();	put();
Kelime (boşluğa, dosya sonuna yada satır sonuna kadar)	>> (yönlendirme operatörü)	<< (yönlendirme)
Satır (dosya sonuna yada satır sonuna kadar)	getline();	<< (yönlendirme)
Nesne (struct veya object tipinde)	read();	write();
İkili dosya tipleri için	Yukarıdakilerin aynısı	Yukarıdakilerin aynısı

Yukarıdaki tablodan okunmak veya yazılmak istenen veri tipine göre bir fonksiyon seçilerek kullanılabilir.

İlave Fonksiyonlar

Aşağıda pekçok zaman dosya işlemlerini kolaylaştıran bazı fonksiyonlar ve kullanım açıklamaları verilmiştir.

İşlem	Fonksiyon	Açıklama
Dosya sonunu kontrol	eof()	Dosyadan okuma yapılırken dosyanın sonuna gelindiğinde true veya 1 değeri döndürür.
Herhangi bir işlemin hatalı olması.	bad()	Yazma veya okuma işlemleri sırasında bir işlemin herhangi bir sebeple gerçekleştirilememesi durumunda true veya 1 döndürür.
Dosya açık mı kontrolü	is_open()	Dosyanın açık olup olmadığını kontrol eder. Şayet açıksa true değilse false döndürür.
Okunan verinin miktarı	gcount();	Dosya açıldıktan sonra şimdiye kadar dosyadan okunmuş olan verinin byte cinsinden değerini döndürür.
Karakter atlatma	igonore()	Verilen miktar kadar karakteri

		atlayarak sonrasından devam eder. Dosyadan okuma işlemi yapılırken belirli bir bilgiyi atlamak için kullanılabilir.
Sıradaki karakteri kontrol	peek();	Sıradaki karakteri kontrol eder ama dosyada ilerlemez. Yani dosyadan okuma işlemi yapıldığında kalınan yerden devam eder ama peek fonksiyonu ile sıradaki karakter kontrol edilebilir.

Rastgele erişim (sadece ikili dosyalarda)	seekg();
	
	seekp();
	
	tellg();
	
	tellp();
		Fonksiyonların sonu g ile bitenler get (Almak) sonu p ile bitenler ise put(Atamak) için kullanılır. Basitçe dosyanın belirli bir konumuna gitmek veya mevcut konumunu öğrenmek için kullanılırlar.

SORU-13: Atomluluk (Atomicity) hakkında bilgi veriniz.

Latince bölünemez anlamına gelen atom kökünden üretilen bu kelime, bilgisayar bilimlerinde çeşitli alanlarda bir bilginin veya bir varlığın bölünemediğini ifade eder.

Örneğin programlama dillerinde bir dilin atomic (bölünemez) en küçük üyesi bu anlama gelmektedir. Mesela C dilinde her satır (statement) atomic (bölünemez) bir varlıktır.

Benzer şekilde bir verinin bölünemezliğini ifade etmek için de veri tabanı, veri güvenliği veya veri iletimi konularında kullanılabilir.

Örneğin veri tabanında bir işlemin (transaction) tamamlanmasının bölünemez olması gerekir. Yani basit bir örnekle bir para transferi bir hesabın değerinin artması ve diğer hesabın değerinin azalmasıdır (havale yapılan kaynak hesaptan havale yapılan hedef hesaba doğru paranın yer değiştirmesi) bu sıradaki işlemlerin bölünmeden tamamlanması (atomic olması) gerekir ve bir hesaptan para eksildikten sonra, diğer hesapa para eklenmeden araya başka işlem giremez.

Benzer şekilde işletim sistemi tasarımı, paralel programlama gibi konularda da bir işlemin atomic olması araya başka işlemlerin girmemesi anlamına gelir.

Örneğin sistem tasarımında kullanılan check and set fonksiyonu önce bir değişkeni kontrol edip sonra değerini değiştirmektedir. Bir değişkenin değeri kontrol edildikten sonra içerisine değer atanmadan farklı işlemler araya girerse bu sırada problem yaşanması mümkündür. Pekçok işlemci tasarımında buna benzer fonksiyonlar sunulmaktadır.

Genel olarak bölünemezlik (atomicity) geliştirilen ortamda daha düşük seviyeli kontroller ile sağlanır. Örneğin işletim sistemlerinde kullanılan semafor'lar (semaphores), kilitler (locks), koşullu değişkenler (conditional variables) ve monitörler (monitors) bunlar örnektir ve işletim sisteminde bir işlemin yapılması öncesinde bölünmezlik sağlayabilirler.

Kullanılan ortama göre farklı yöntemlerle benzer bölünmezlikler geliştirilebilir. Örneğin veritabanı programlama sırasında koşul (condition) veya kilit (lock) kullanımı bölünmezliği sağlayabilir.

SORU-14: Gizli Dosya (Hidden File) hakkında bilgi veriniz.

İşletim sistemlerinde kullanılan dosya tiplerinden birisidir. Basitçe sistemde kullanılan kritik dosyaların kullanıcı müdahalesinden korumak için geliştirilmiştir. Örneğin Windows™ işletim sisteminde kullanılan gizli dosyaların ağırlıklı amacı sistem dosyalarını ve önemli ayarlamaları içeren klasörleri korumaktır.

Linux / Unix gibi işletim sistemlerinde de gizli dosyaların isimleri "." işareti ile başlamaktadır. İşletim sistemi otomatik olarak bu dosyaları gizli dosya adleder ve klasik dosya görüntüleme ve dosya sistemi dolaşma işlemlerinde bu dosyalar gizlenir.

Örneğin linux işletim sisteminde kullanılan ve dosya listelemeye yarayan "ls" komutunda gizli dosyaları göstermek için -h (hidden) parametresi verilmesi gerekir.

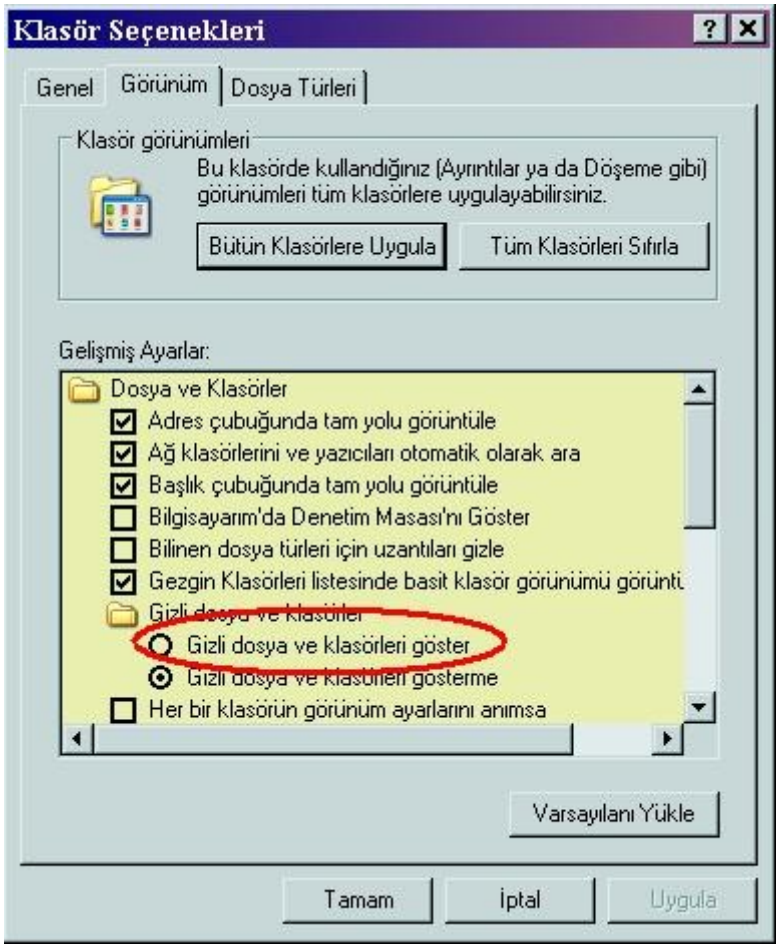
"ls -h" şeklinde. Arşiv dosyalarının da görülmesi için "ls -ah" şeklinde kullanılabilir.

DOS işletim sisteminde bu işlemi yapan "dir" komutunu da /h parametresi ile kullanabiliriz.

"dir /ah" şeklinde çağrılabilir ve görüntülenebilir.

Windows işletim sisteminde ise gizli dosyaların gösterilmesi için "klasör seçeneklerinden" gerekli ayarların yapılması gerekir.

Basitçe "bilgisayarım>Araçlar>Klasör Seçenekleri>Görünüm" ekranında aşağıdaki seçeneğin seçilmesi gizli dosyaların gösterilmesi için yeterlidir:



Ayrıca windows ve DOS işletim sistemlerinde dosyaların gizli veya arşiv dosyası olduğunu görmek veya tiplerini değiştirmek için "attrib" komutu kullanılabilir. Basitçe + ve – işaretleri ile dosyaya özellik eklenebilir veya çıkarılabilir. Örneğin bir dosyayı gizli dosya yapabilmek için

"attrib +h dosya" şeklinde komutu kullanmak gerekir.

Linux / Unix işletim sistemlerinde ise dosyanın isminin başına nokta "." işaretinin eklenmesi yeterlidir.

SORU-15: Sonda (Probe) hakkında bilgi veriniz.

Veri iletişimi sırasında veriye erişmek için yapılan her bir erişim işlemine verilen isimdir. Örneğin veri diskte veya hafızada duruyor olsun, veriye erişmek için yapılan her bir hafıza veya disk erişimine sonda ismi verilir.

Basit bir dizide veriyi aradığımızı düşünelim. Örneğin dizimiz:

int a[] = { 2 ,3 ,8 ,7};

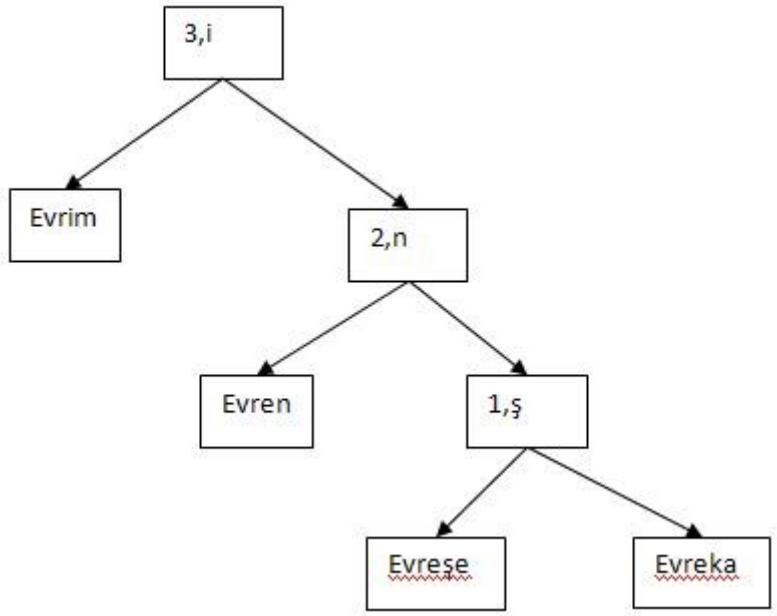
olarak verilmiş olsun. Örneğin doğrusal arama (linear search) kullanarak dizinin ilk elemanından son elemanına kadar diziyi taradığımızı düşünelim. 8 sayısını bulmak için sırasıyla 2,3 ve 8 sayıları diziden okunmalıdır. Bu durumda toplam 3 sonda işlemi (probing) yapılmış olur.

SORU-16: Patricia ağacı (PATRICIA Tree) hakkında bilgi veriniz.

Bilgisayar bilimlerinde sıkça kullanılan TRIE ağacının özel bir hali olan patricia ağacında genellikle sözlüksel olarak (lexiconically) veriler tutulur. Radix ağacı (radix tree) ve farklı ikil ağacı (crit bit tree) ile oldukça benzer olan patricia ağacının, TRIE ağacından en büyük farkı tutulan verilerin ortak olan noktalarından sonra farklılaşılan yönlerine göre dallanma olmasıdır.

Aşağıda verilen kelimelerin ağaçta tutulmaları gösterilmiştir:

- Evren
- Evreşe
- Evreka
- Evrensel



Yukarıdaki şekilde de gösterildiği üzere ağacın dallanmaları verilen kelimelerin birbiri ile farklılaştıkları noktalarda olmaktadır.

SORU-17: Brent Yöntemi (Brent's Method) hakkında bilgi veriniz.

Bilgisayar bilimlerinde dosya yönetiminde özetleme (hashing) için kullanılan bir yöntemdir. Bu yönteme göre ekleme sırasında bazı değişiklikler ile yerleştirilen kayıtların arama hızını arttırmak ön plandadır. Özet tabloya (hash table) yerleştirilen bir kayıdın çeşitli durumlarda yeri değiştirilerek okuma zamanının arttırılması hedeflenir. İki ayrı zincir tutmaktadır.

- Birincil sonda zincirinde (primary probe chain) sayının aranması ve yerleştirilmesi için gereken sıra tutulur.
- İkincil sonda zincirinde (secondary probe chain) ise sayının yer değiştirmesi için gereken sıra tutulur.

Çalışmasını daha iyi anlamak için aşağıdaki örnek üzerinden yöntemi anlamaya çalışalım:

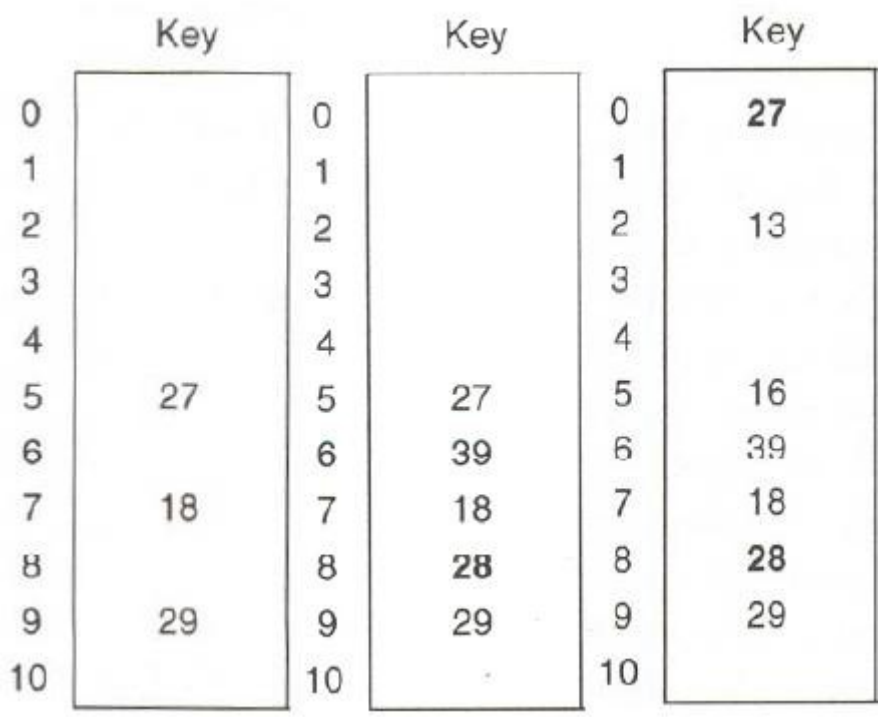
Eklenmek istenen sayılar : 27, 18, 29, 28, 39, 13, 16 olsun.

Ekleme sırasında kullanılacak olan öze fonksiyonlarımız (hashing function):

H1(anahtar) = anahtar mod 11 (anahtarın 11′e bölümünden kalan değer)

H2(anahtar) = bölüm (anahtar/11) mod 11 (anahtarın 11′e bölüm değerinin tekrar 11′e bölümünden kalan değer)

İlk 3 sayı herhangi bir çakışma olmadan özet tablomuza koyulabilirler. Bu sayılardan sonra tablomuz:



Yukarıdaki şekilde olur. Burada koyu renkle gösterilen anahtarlar değiştirilmiştir.

Örneğin ilk durumda 27, 18 ve 29 sayıları problemsiz bir şekilde koyulur. Ardından gelen 28 ve 39 sayısı çakışır (collision). İkisi de 6 adresine konulmalıyken sırasıyla önce 28 daha sonra da 39 bu adrese konulur. En son durumda 6. adreste 39 bulunur. 28 değeri ise ikinci özetleme fonksiyonuna (hashing function sokulur. H2(28) = 2 olduğu için 6+2 = 8. adrese 28 değeri taşınır.

Ardından gelen 13 problemsiz bir şekilde 2. adrese koyulurken, 16 sayısı için yine çakışma söz konusudur.

16 sayısı 5. adree koyulduktan sonra bu adreste bulunan 27 sayısı taşınmaya başlanır. H2(27) = 2 olduğu için 2şer atlanarak uygun yer aranır. Önce 5+2 = 7 kontrol edilir, dolu olduğu için 5+2+2 = 9 kontrol edilir dolu olduğu için 5+2+2+2 = 11 mod 11 = 0 kontrol edilir ve boş bulunduğu için bu adrese sayı yerleştirilir.

Görüldüğü üzere brent metodunda (brents method) her zaman son gelen değer adreste tutulmakta ve adreste bulunan eski değer hareket ettirilmektedir. Bu anlamda dinamik metot olarak sınıflandırılabilecek brent yönteminde, EISCH yöntemi, LISCH yöntemi veya doğrusal sondalama (linear probing) yöntemlerinden farklı olarak sayılar ilk konuldukları yerden farklı yerlere taşınabilirler.

SORU-18: B Ağacı (B-Tree) hakkında bilgi veriniz.

İçerik

İsminin nereden geldiği (B harfinin) tartışmalı olduğu bu ağaç yapısındaki amaç arama zamanını kısaltmaktır. Buna göre ağacın her düğümünde belirli sayıda anahtar veya kayıt tutularak arama işleminin hızlandırılması öngörülmüştür.

Arama hızının artmasına karşılık silme ve ekleme işlemlerinin nispeten yavaşlaması söz konusudur.

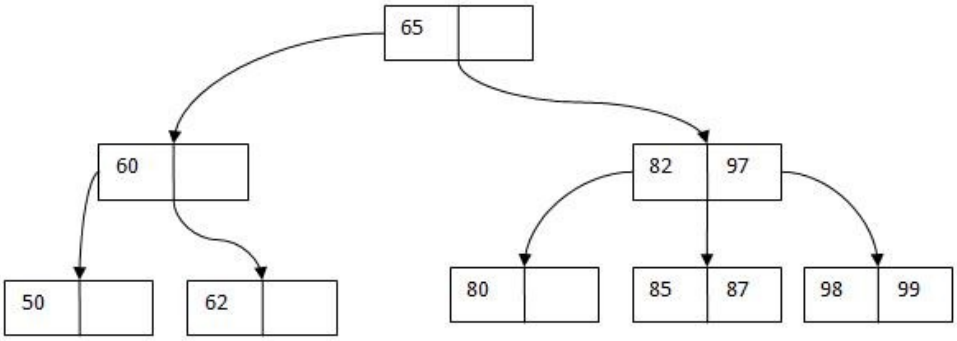
1. B-Ağacının tanımı

Bir B-Ağacı (B-Tree) aşağıdaki özelliklere sahip olmalıdır:

- Her düğümün (node) en fazla m çocuğu bulunmalıdır. (Bu sayının üzerinde eleman bulunursa düğümün çoğaltılması gerekir)
- Kök (root) ve yaprak (leaf) düğümleri haricindeki her düğümün en az m/2 adet elemanı bulunmalıdır. (Bu sayının altında eleman bulunursa düğüm kaldırılır)
- Bütün yapraklar aynı seviyede olmak zorundadır. Bir yaprağın seviyesinin düşmesi durumunda (daha yukarı çıkması veya daha sığ olması durumunda) ağaçta yapısal değişiklik gerekir.
- Herhangi bir düğümde k çocuk bulunuyorsa k-1 elemanı gösteren anahtar (key) bulunmalıdır.

2. Örnek B-Ağacı

Aşağıda örnek bir b ağacı gösterilmiştir:



Aşağıda b ağacı üzerinde yapılan ekleme, silme ve arama işlemleri açıklanmıştır.

3. B Ağacında Arama

Bağacında (Btree) arama işlemi kökten başlar. Aranan sayı kök düğümde bulunamaması halinde arama işlemi kökte bulunan anahtarların sağında solunda veya arasında şeklinde yönlendirilir. Örneğin yukarıdaki B-ağacında 87 anahtarı aranıyor olsun. Arama işlemi için aşağıdaki adımlar gerekir:

1. kök düğüme bakılır. 87 değeri 65′ten büyüktür. Kök düğümde tek anahtar olduğu için 65′in sağındaki gösterici(pointer) takip edilir.

2. 65. sağındaki düğüme gidilir ve ilk anahtar olan 82 ile aranan anahtar olan 87 karşılaştırılır. 87 değeri 82′den büyüktür. Öyleyse ikinci anahtar olan 97 ile karşılaştırılır. 87 bu değerden küçük olduğu için bu düğümde 82 ile 97 arasında bulunan gösterici izlenir.

3. Son olarak 82 ile 97 arasındaki düğüm izlenerek ulaşılan düğümdeki anahtar ile 87 karşılaştırılır. Bu düğümdeki ilk anahtar 85′tir. 87 bu değerden büyüktür. Düğümdeki bir sonraki anahtar alınır ve 87 değeri bulunur.

B-ağaçlarının bir özelliği ağacın her düğümündeki anahtarların sıralı oluşudur. Bu yüzden bir düğümde istenen anahtar aranırken, düğümde bulunan sayılara teker teker bakılır (linear search, doğrusal arama)

4. B Ağacına Ekleme

B ağaçlarında veri yaprak düğümlerden gösterilir (pointer). Yani aslında veri ağacın düğümlerinde değil son düğümlerden gösterilen hafıza bölmeleri (RAM veya Dosya) olarak tutulur. Dolayısıyla B ağacında ekleme işlemi sırasında anahtarlar üzerinden arama ve değişiklikler yapılır ve nihayetinde amaç B ağacının son düğümleri olan yaprak düğümlerden veriyi göstermektir.

B ağaçlarında veri eklemek için aşağıdaki adımlar takip edilir:

1. Ağaçta eklenecek doğru yaprak düğümü aranır. (Bu işlem için bir önceki adımda anlatılan arama algoritması kullanılır)

2. Şayet bulunan yaprak düğümde azami anahtar sayısından (maximum key number) daha az eleman varsa (yani anahtar eklemek için boş yer varsa) bu düğüme ekleme işlemi yapılır.

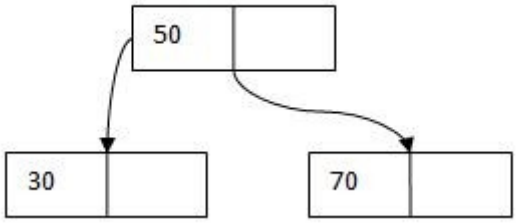
3. Şayet yeterli yer yoksa bu durumda bulunan bu yapra düğüm iki düğüme bölünür ve aşağıdaki adımlar izlenir:

1. Yeni eleman eklendikten sonra düğümde bulunan anahtarlar sıralanır ve ortadaki elemandan bölünür. (median değeri bulunur)

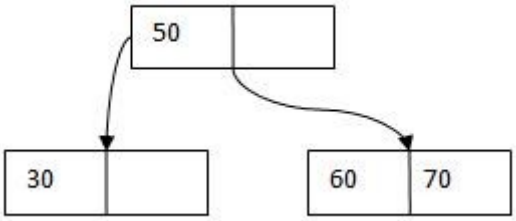
2. Ortanca değerden büyük elemanlar yeni oluşturulan sağ düğüme ve küçük elemanlar da sol düğüme konulur.

3. Ortanca eleman (median) ise bir üst düğüme konulur.

Yukarıdaki ekleme işlemini aşağıdaki örnek ağaç üzerinden görelim.

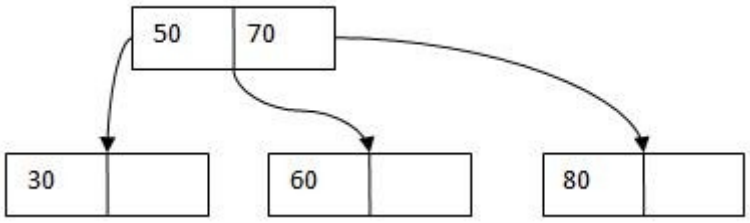


Örneğin azami anahtar sayısıs 2 olan yukarıdaki örnek ağaçta ekleme işlemi yapalım ve değer olarak 60 ekleyelim:



Yukarıdaki ekleme işlemi, ekleme algoritmamızdaki 2. durumda gerçekleşmektedir. Yani anahtarımızın ekleneceği yaprakta boş yer bulunmaktadır ve buraya yeni anahtarı ekleriz.

Şayet yukarıdaki ağaca 80 değerini ekleyecek olsaydık bu durumda da algoritmamızdaki 3. ihtimal gerçekleşmiş olacaktı.



Görüldüğü üzere 80 anahtarının ekleneceği düğüm dolmuş ve azami 2 anahtar olamsı gerekirken 3 anahtar olmuştur. Ortanca değer (median) 70 olan bu ekleme işleminden sonra ortanca değer bir üst düğüme çıkmış ve iki farklı düğüme ortanca elemanın solundaki ve sağındaki değerler yukarıdaki şekilde dağıtılmıştır.

5. B Ağacından Silme

B ağacı yukarıdaki özellikleri bölümünde anlatılan özelliklerin bozulmaması için silme işlemi sırasında aşağıdaki iki yöntemden birisini izler:

1. çözümde ağaçtan ilgili anahtar bulunup silinir ve bütün ağaç yeniden inşa edilir

2. çözümde ağaçtan ilgili anahtar bulunup silinir ve bulma işlemi sırasında geçilen ağacın kısımları yeniden inşa edilir.

Ayrıca B+ ağacı (B plus tree) ve B# ağacı (B number tree) şeklinde alt çeşitleri de bulunmaktadır.

Ayrıca B ağacının özel birer uygulaması olarak aşağıdaki ağaçlara bakabilirsiniz:

2-3 Ağacı

2-3-4 Ağacı

SORU-19: Dizgi (String) hakkında bilgi veriniz.

Bir dilde bulunan ve o dilin tanımlı olan alfabesi içerisindeki sembollerin çeşitli sayılarda ve çeşitli sırada dizilmesi ile elde edilen yazılardır.

Örneğin bir dildeki alfabe aşağıdaki şekilde tanımlı olsun:

$\sum_1 = \{0,1\}$

Buna göre dilimizde sadece "0″ ve "1″ sembolleri tanımlı demektir. Bu dilde örneğin w_1=0 veya w_2=10101011010 gibi bir dizgi elde etmek mümkündür.

Bir dizginin belirli bir kımını içeren dizgiye ise alt dizgi adı verilir. Örneğin w_3=1011 dizgisi w_2 dizgisinin bir altdizgisidir.

Ayrıca iki dizginin arka arkaya eklenmesine de üleştirme(concatenation) denilir.Örneğin w_1 ile w_3 dizgilerinin üleştirilmiş hali w_4=01011 olur.

Dizgiler ile ilgili diğer yazılar:

- Dizgi Parçalayıcısı (String Tokenizer)
- Dizgi Karşılaştırma (String Comparison)
- Veri Tabanı Dizgi işlemleri (Database String manipulations)
- En uzun ortak küme (longest common subsequence) problemi
- Dizgi Eş şekilliliği (String Isomorphism)
- C dilinde dizgi kopyalama (strcpy)

- Alt Dizgi (Substring)
- Dizgi Hizalama (String Alignment) Problemi

Dizgi Karşılaştırma/Arama/Mesafe algoritmaları:

- Boyer Moore Dizgi Karşılaştırması (Boyer Moore String Comparison Algorithm)
- Knuth Moris Prat Dizgi Arama Algoritması (Knuth Morris Prat Algorithm)
- Dizgi Eş şekilliliği (String Isomorphism)
- Needleman Wunsch Yaslama Algoritması (String Alignment)
- Smith Waterman Yaslama Algoritması (String Alignment)
- Hunt Macllor Yaslama Algoritması (String Alignment)
- Horspool Algoritması (Dizgi arama algoritması)
- Levenstein Mesafesi (Levenshtein Distance)
- Hamming Mesafesi (Hamming Distance)
- Diff komutu
- Jaccard Indeksi, mesafesi ve katsayısı (Jaccard Index)
- Dice Sorensen Benzerliği (Dice Sorensen Similarity)

Dizgi Parçalama (Parsing) algoritmaları:

- LL(1) parçalama algoritması
- SLR(1) parçalama algoritması
- Earley Parçalama Algoritması

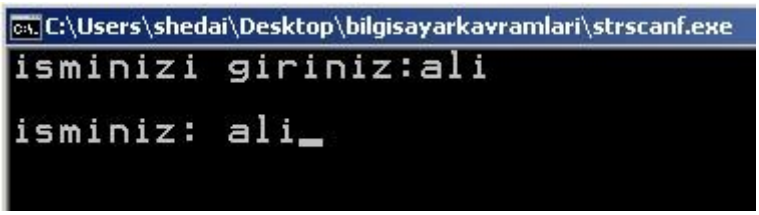
C ile dizgi okuma

C dilinde klavyeden dizgi (String) okumak için kullanılan en basit fonksiyon scanf fonksiyonudur. Bu fonksiyonu basit bir uygulamada aşağıdaki şekilde kullanabiliriz:

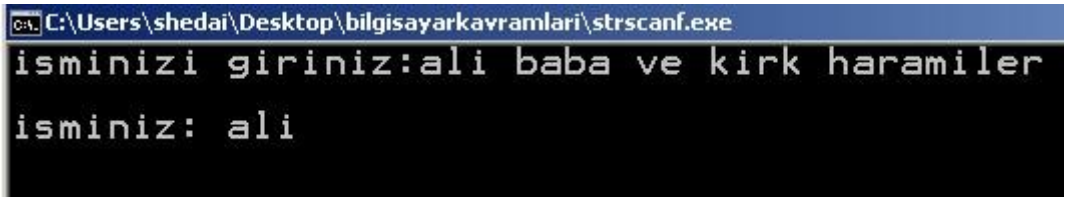
```
1   #include <stdio.h>
2   #include <conio.h>
3   //www.bilgisayarkavramlari.com
4   int main(){
5       char isim[100];
6       printf("isminizi giriniz:");
7       scanf("%s",isim);
8       printf("\nisminiz: %s",isim);
9       getch();
10  }
```

Yukarıdaki kodda, 5. satırda tanımlalan karakter dizisi (string) içerisine 7. satırda %s parametresi ile scanf fonksiyonu kullanılarak bir dizgi okunmuştur. Bu dizginin içeriği kodun 8. satırında ekrana basılmıştır.

Örneğin yukarıdaki kod aşağıdaki şekilde çalıştırılabilir:



Görüldüğü üzere, kullanıcı isim olarak "ali" girmiş ve ekranda, girdiği bu dizgiyi görmüştür. Ancak aynı kodu çalıştırarak aşağıdaki şekilde bir dizgi girilirse problem yaşanır:



Yukarıdaki girdide "ali baba ve kirk haramiler" şeklinde 5 kelimeden oluşan bir dizgi girilirken, bu dizginin sadece ilk kelimesi scanf tarafından okunmuştur. Aslında burada bir hata yoktur çünkü scanf fonksiyonu, boşluk karakteri veya satır sonu gibi karakterlere kadar olan dizgileri okur. Yukarıdaki gibi birden fazla kelimeden oluşan dizgiler okunmak istendiğinde, aşağıdaki kodda da gösterilen gets fonksiyonu kullanılabilir:

```
1   #include <stdio.h>
2   #include <conio.h>
3   //www.bilgisayarkavramlari.com
4   int main(){
5       char isim[100];
6       printf("isminizi giriniz:");
7       gets(isim);
8       printf("\nisminiz: %s",isim);
9       getch();
10  }
```

Yukarıdaki kodda bir önceki koda göre sadece 7. satırda bulunan scanf fonksiyonu, gets fonksiyonu ile değiştirilmiştir. Kodumuzun yeni halini çalıştırdığımızda, aşağıdaki şekilde birden fazla kelime okuyabildiğimizi görürüz:

Dizgilerin Eşitliği

İki farklı değişkende bulunan dizginin eşit olup olmadığı, programlama dillerinde bulunan klasik operatörler ile yapılamaz.

Örneğin aşağıdaki kodlama C dili açısından doğru olsa bile mantıksal olarak hatalıdır (logic error):

char a[100] = “bilgisayarkavramlari.com”;

char b[100]= “bilgisayarkavramlari.com”;

if(a==b)

Yukarıdaki kodun, son satırında bulunan eşitlik kontrolü, C dili açısından ne derleme (compile) ne de çalışma (run-time) hatası döndürmez. Ancak buradaki karşılaştırma aslında iki dizinin (array) hafızada (RAM) aynı yeri gösterip göstermediğini sorgulamaktadır.

Yukarıdaki kod, her zaman için yanlış (false) döner ve if kontrolüne hiçbir zaman girilemez. Bunun yerine dizgilerin içeriklerinin karakter karakter kontrol edilmesi ve dizgilerin boyutlarının eşit olup olmadığının sorgulanması gerekir.. Bizim iki dizginin eşitliğinden anladığımız genelde budur. Bu kontrole literatürde derin karşılaştırma (deep compare) ismi verilmektedir.. Klasik olarak yapılan a==b kontrolü ise sığ karşılaştırma (shallow compare) olarak geçmektedir.

Derin karşılaştırma için kendimiz bir fonksiyon yazabileceğimiz gibi, C dilinde var olan string.h kütüphanesindeki strcmp fonksiyonunu kullanabiliriz. Bu fonksiyon iki dizgiyi (string) sözlük sıralamasına göre karşılaştırır (lexiconically) ve şayet eşitlerse 0 değerini döndürür.

C dilinde 0 değeri mantıksal olarak yanlış (false) olduğu için eşit olmaları durumunda bir if bloğunun çalışmasını istiyorsak, aşağıdaki şekilde yazabiliriz:

if(!strcmp(a,b))

yukarıdaki kontrol, a ve b dizgilerinin eşitliği durumunda geçer.

Aynı kontrol JAVA veya C# gibi dillerde, String sınıfının bulunması sayesinde, ilave bir kütüphane ve fonksiyona gerek kalmadan çözülebilir.

Örneğin JAVA dili için:

String a= “www.bilgisayarkavramlari.com”;

String b= “www.bilgisayarkavramlari.com”;

if(a.equals(b))

kontrolü yapılması yeterlidir. Java dilinde, bulunan ve dizgileri karşılaştırmak için kullanılan equals fonksiyonu C# dilinde ilk harfi büyük olarak Equals şeklinde yazılarak çalıştırılabilir (burada bir kere daha JAVA’yı taklit ederken, java kodlarının çalışmaması için özel gayret sarf eden microsoft geliştiricilerini selamlıyoruz.)

Dizgilerin birbirine eklenmesi (concatenate, üleştirme) için C ve C++ gibi dillerde strcat fonksiyonu kullanılabilir.

Örneğin:

char a[100]= “www.”;

char b[100] = “bilgisayarkavramlari.com”;

strcat(a,b);

printf(“%s”,a);

şeklindeki bir kod, ekrana “www.bilgisayarkavramlari.com” sonucunu basacaktır.

Aynı üleştirme işlemi, JAVA veya C# için basit bir toplama (+) işlemi ile yapılabilir.

String a=”www.”;

String b=”bilgisayarkavramlari.com”;

System.out.println(a+b);

yukarıdaki kod, ekrana “www.bilgisayarkavramlari.com” sonucunu basarken aynı kodu C# dilinde sadece son satırını Console.write(a+b) olarak değiştirerek deneyebilirsiniz.

SORU-20: Özetleme Fonksiyonları (Hash Function) hakkında bilgi veriniz.

Özetleme fonksiyonlarının çalışma şekli, uzun bir girdiyi alarak daha kısa bir alanda göstermektir. Amaç girende bir değişiklik olduğunda bunun çıkışa da yansımasıdır.

Buna göre özetleme fonksiyonları ya veri güvenliğinde, verinin farklı olup olmadığını kontrol etmeye yarar ya da verileri sınıflandırmak için kullanılır.

Anlaşılması en basit özetleme fonksiyonu modülo işlemidir. Buna göre örneğin mod 10 işlemini ele alalım, aşağıdaki sayıların mod 10 sonuçları listelenmiş ve gruplanmıştır:

Sayılar: 8,3 ,4,12,432,34,95,344,549,389,2339,349,54,81,17,62,94,67,44,9

Demet (Buket, Bucket)	Sayılar					
0						
1	81					
2	12	432	62			
3	3					
4	4	34	344	54	94	44
5	95					
6						
7	17	67				
8	8					
9	389	2339	349	9		

Kısaca yukarıdaki sayıların hepsi 1 haneli bir sayıya özetlenmiştir. Örneğin 81 -> 1, 344 -> 4 gibi. Elbette aynı sayıya özetlenen birden fazla sayı bulunmaktadır. Bu duruma çakışma (collusion) adı verilmektedir.

Özeteleme fonksiyonlarının ingilizcesi olan Hash kelimesinin kökü arapçadan girmiş olan haşhaş kelimesi ile aynıdır. Ve insan üzerinde yapmış olduğu deformasyondan esinlenerek hash function'a giren bilgilere yapmış olduğu deformasyondan dolayı bu ismi almıştır.