DERS 7

Bugün **hashing** yani **kıyım fonksiyonu** konusunda iki derslik bir dizi başlıyor; bu önemli bir teknik ve birçok yerde karşımıza çıkıyor. Kıyım fonksiyonunu, compileryada derleyicilerde ortaya çıkan ve **sembol tablosu problemi** denilen bir sorunu örnek olarak anlatacağız. Elimizde içinde n adet kayıt olan bir S tablosu olduğunu düşünün. Burada biraz daha açık olacağımvebu kayda x diyelim. x genellikle gerçek veriyi gösteren bir işaretçidir ve x kaydından söz ettiğimiz zamanaslında verinin yerini belirten bir göstericiden bahsederiz. Veri yada kayıttan bahsedince de, bu kaydın içinde x'in anahtarı denen bir anahtar mevcut.

Bazı dillerde buna anahtar, bazılarında x nokta anahtar x dotkey, bazılarında da x oku anahtarı denir; değişik dillerde değişik terimler kullanılabilir. Ve genellikle bu anahtarla ilgili satellitedata yada uydu verisi denen ek bir bilgi vardır ve bu bilgi anahtarla birlikte hareket eder. Bu sıralamada da geçerlidir ama sıralamada kayıtları sıralarsınız; bağımsız anahtarları değil. Burada tablonun içinde veriler üzerinde bazı işlemler yapmayı düşünüyoruz.

Bu tabloya bir yeni xöğesinieklemeyi amaçlıyoruz, yani aslında x elemanını ekleyerek tabloyu güncelleştireceğiz. Tablodan bir öğeyi desilebilmeyi istiyoruz. Yani x öğesini tablodan çıkarmak istiyoruz. Ve belirli bir anahtarı da arayabilmek istiyoruz. Böylece bu, x'in anahtarı k'ya eşit olduğunda bize x değerini, böyle bir x yoksa da nil yani boş değerini versin. Kısacası tabloya öğeler ekleyebilmek yada çıkarabilmek, aynı zamanda da belirli bir anahtarı olan bir öğe olup olmadığına bakmak istiyoruz. Burada "delete" yani silme işleminde anahtar gerekmediğini not ediniz..Silmede sadece kayıt gerekli; yani belirli bir anahtarın bir şeyini silmek isterseniz ve onun bir işaretçisi yoksa, bu durumda önce arama yapmalı sonra onu silmelisiniz.

Elinizde bir işlem kümesi varsa ve örnekteki silme işlemi gibi işlemler kümeyi değiştiriyorsa, bu tür kümelere **dinamik kümeler** denir. Bunun gibi işlemler kümeyi dinamik yaparlar, zaman içinde değişirler. Bazen sabit bir veri yapısı oluşturmak istersiniz. Bu statik bir kümeyada ünite olacaktır. Bununla yapacaklarınız bir şeylere bakmak olacaktır.Ama biz programcılıkta kümelerin genelde dinamik olmasını isteriz. Bu kümelere elemanlar eklemek yada çıkartmak isteriz ve bunların dışında kümeyi değiştiren, kümenin üyelik yapısını değiştiren işlemleri de yapmak isteriz. Bu işlemlerin en basit uygulaması bazen gözden kaçırılır.

İnsanların bu basit veri yapısı ile çalışmak yerine, sıklıkla daha karmaşık veri yapıları kullanmaları beni şaşırtıyor. Bunun adı "directaccesstable" yada**doğrudan erişim tablosu**. Her zaman çalışmaz. Size çalışması için gerekli koşulları vereceğim. Anahtarları küçük dağılımınızdan alırsanız çalışır. Bu nedenle anahtarların m elemanı olan U setinden alındığını varsayın; 0'dan m-1'e kadarm elemanı var. Ve anahtarların da farklı olduklarını varsayacağız.

Doğrudan erişim tablosunun yada çizelgesinin çalışması için;
T[0…m-1] dizilimini oluşturursunuz. Bu dizilim S dinamik kümesini temsil eder, öyle ki,
x kümenin içindeyse ve k key[x]'e eşitse T(k) = x olur. Diğer durumlarda ise T(k) = nil olur. Yani elinizde bir dizilim var. Bu dizilimdeki kayıtlardan birinin anahtarının değeri k ise, örneğin 15, 15'inci yuvada bu eleman bulunur. Bu değer kümenin içinde değilse, yuvada nil değeri vardır.

Bu çok basit bir veri yapısıdır. Eklemek için, ekleme yapacağınız yere gidin ve değeri eklenecek değerle değiştirin. Silmek için mevcut değeri oradan çıkarın. Aramak için de dizinleyin ve o alanda ne olduğuna bakın. Gördüğünüz gibi çok basit bir veri yapısı var. Bu nedenle de bu işlemlerin hepsi en kötü durumda sabit zaman alır. Ama pratik açıdan bu stratejiyi kullanabileceğiniz durumlar oldukça sınırlıdır. Buradaki sınırlamanın temelinde acaba ne vardır?

Evet?EvetBu bir sınırlama. Ama bundan daha ciddi bir sınırlama var. Evet? çizmesi mi zor diyorsunuz Bu ne anlama geliyor,?Evet amam-1 çok büyük bir sayı olabilir. Örneğin, kümemin 64 bitlik değerler içermesi, yani tablomda 64 bitlik sayıları depolama durumunda. Belki elimizde küçük bir küme var, yani bu elemanlardan sadece birkaç bin tane var.

Ama bu elemanların her biri 64 bitlik. Bu durumda bu strateji, 0'dan 2'nin 64 -1'inci kuvvetine kadar olan değerleri içeren bir dizilimi gerektirir. 2^{64-1}'in büyüklüğü nedir? Yaklaşık olarak -- bayağı büyük. 18 quintilyon dolaylarında. Yani zilyonlarca demek istiyorum, güncel hayatta kullandığımız –ilyonlar gibi değil. Milyar yada trilyon değil. 18 quintilyon. Tamam, bu gerçekten de çok büyük bir sayı. Bundan daha da kötüsü anahtarların insanların isimleri gibi harf dizgilerinden alındığında ortaya çıkar. O zaman bu strateji hiç iyi bir uygulama yöntemi olmaz çünkü saklamak istediğiniz makul boyuttaki değer kümeleri için bile tablonun çoğu boş kalacaktır.

Bu nedenle düşüncemiz tablonun özelliklerinden bazılarını koruyup boyutunu küçük tutmaya çalışmak olacaktır. Burada devreye kıyım fonksiyonu yani hash fonksiyonu girer. Kıyım fonksiyonu uygulamasında H fonksiyonunu kullanarak anahtarları tablo T'nin içindeki yuvalara rastgele eşlemleriz, yani "mapping" işlemini uygularız; burada" rastgele'yi tırnak içine aldım çünkü aslında eşlemleme tamamen rastgele değil. Burada her dizilim indeksine yuva yani slot diyeceğiz. Böylece bunun büyük bir tablo olduğunu düşünebilirsiniz ve tablonun içinde de değerleri depoladığınız yuvalar var.

Bu nedenle elimizde bir anahtarlar evreni oluşabilir. Buna U diyelim. Ve burada da tablomuz var; tablonun m adet yuvası var. Pratikte elimizde bir S kümesi vardır ve bu kümeevrenin çok küçük bir parçasıdır. Sonra yapacağımız işlem S' den bir elemanı alıp örneğin şuraya eşlemlemektir; ardından bir başka eleman seçeriz ve bu elemana kıyım fonksiyonunu uygularız. Kıyım fonksiyonunun yapacağı şey T'nin içindeki bir yuvayı belirlemektir.

Biri şuraya gidebilir; başka biri aşağıda buraya gidebilir. Bu şekilde elemanların tablo içinde dağıtımını yapar. Pekiyi bu işlemi yaparken karşılaşılabilecek sorun nedir? Şu ana kadar biraz şanslıydım. Buradaki potansiyel problem ne olabilir? Evet, S'nin içindeki iki değerin aynı yuvaya atanması.. Yani buradaki bir eleman, daha önce başka bir elemanın eşlemlendiği yuvaya eşlemlenebilir. Bu durumun oluşması haline "collision" yani çarpışma deriz.

Buradaki değerleri küçük bir kümeye eşlemlemeye çalışıyoruz ama buradaki değerlerden çoğunu eşlemeye kalktığımızda eşlemlememizde şanssız olabiliriz; elemanlar tabloya sığmayabilir. Bu durumda eklenecek kayıt dolu bir yuvaya eşlendiği takdirde çarpışma dediğimiz yani collision dediğimiz olay ortaya çıkar. İlk bakışta bu yöntemiyi değilmiş gibi görünüyor. Ama hayır, yapabileceğimiz çok basit bir şey var. İki ayrı şey aynı yuvaya eşlenirse ne yapabiliriz? Burada tüm kümenin temsilini istiyoruz; yani önbellek uygulamalarında olduğu gibi veri kaybetme lüksümüz yok. Önbellekler de kıyım fonksiyonunu kullanırlar ama önbellekte kümeleri tümüylekoruma kaygısı olmadığından ihtiyaç duyulmayan veriler atılır.

Ama programlama yaptığınız bir kıyım tablosunda elinizdeki değerlerin kümelerin içindekilerle bire bir aynı olmasını istersiniz çünkü bir şeyin o kümeye ait olup olmadığını bilmeniz gerekiyor. Öyleyse buradaki iyi strateji ne olabilir? Evet, her yuva için bir liste oluşturmak ve yuvaya eşlemlenen her elemanı listelemek ve buna "resolvingcollisionsbychaining" yani **çarpışmaları zincirlemeyle çözmek** diyoruz. Ve buradaki yaklaşım aynı yuvadaki kayıtları bir liste ile ilişkilendirmektir.

Örneğin, bunun kıyımlama tablom olduğunu ve bunun da i yuvası olduğunu düşünün. Burada birkaç değer olabilir, bu nedenle anahtar değerlerini de belirteceğim. S'nin elemanlarından birkaçı bu tabloya eklenmiş olabilir. Yapacağım şey bunları birbirine gördüğünüz gibi bağlamaktır. Tamam, burada nil göstergesi var. Bu bölge uydu verisi, bunlar da anahtarlardır. Eğer bunlar i yuvasında birbirine bağlıysa, 49'a uygulanan kıyımlama fonksiyonu, 86'ya

uygulanan kıyımlama fonksiyonuna ve 52'ye uygulanan kıyımlama fonksiyonuna eşit olmak zorundadır; buneye eşittir?

Belirtmeyi unuttuğum bir şey var. i. Güzel. Bunu tam olarak anlamasanız da testlerdeki ustalığınız size hatırlatmalıydı: "i'yi yazmayı unuttu" demeliydiniz. Şu i'ye eşittir. Yani 49'un kıyım fonksiyonu tabloda, mesela i değerli bir dizin yaratır ve o yere yönelik her fonksiyon listede birbirine bağlanır; tamam mı? Kayıtların hepsi.. Bunun mekaniği konusunda sorular var mı? Umarım ki hepiniz temel kıyım fonksiyonunu 6.001'de görmüşsünüzdür. Başka bir derste mi öğretiyorlar? Eskiden mi bu derste veriyorlardı? Tamam, bazılarınız belki diyor. Eskiden öğretiyorlardı. İyi, şimdi bu stratejiyi çözümleyelim.

Çözümleme: Önce en kötü durumu ele alalım. Kıyım fonksiyonu uygulamasında en kötü durumda acaba ne olur? Evet elinizi kaldırın ki size söz verebileyim. Evet? Kıyım fonksiyonu seçtiğim S kümesinin içindeki bütün kıyım anahtarlarını aynı değere eşlemler. Bu kötü olur; çünküher anahtar aynı yuvaya eşlemlenmiştir. Bunun olması halinde bu veri yapısını korumak için gösterişli bir liste elde ederim. Bu durumda da tabloların, kıyım fonksiyonunun ve benzerlerinin anlamı kalmaz; Elimde kocaman bir liste olur. Pekiyi, bu durumda erişim ne kadar sürer? Bu listeye bir şey eklemek, daha da önemlisi bir arama yapmak, bir şeyin listede olduğunu kontrol etmek en kötü durumda ne kadar zamanımı alır?

Evet n düzeyinde zaman alır. Çünkü bunlar sadece linkler yada bağlantılardır ve elimizde bir tek bağlantı listesi vardır. Yani, S'nin boyutunun n'ye eşit olduğunu kabul edersek, erişim Θ(n) süresini alır. Sonuçta en kötü durum açısından bu çok çekici gelmiyor. Ve biz bu tür problemlerde çok iyi sonuç alınacak veri yapılarını göreceğiz. Ama onlar da kıyım fonksiyonunun ortalama durumunda çok iyi sonuç vermiyorlar. Haydişimdi ortalama durumu çözümleyelim.

Ortalama durum çözümlemesi için; elinizde ne zaman bir ortalama yada olasılık durumu varsa,varsayımlarınızı belirtmeniz gerekiyor. Sistemin davranışı ile ilgili varsayımınızı söylemeniz gerek. Ve bunu yapmak da çok zordur çünkü kıyım fonksiyonunun ne olduğunu bilemeyebilirsiniz. Şimdilik ideal bir kıyım fonksiyonu hayal edelim. İdeal bir kıyım fonksiyonu ne yapmalı?

Evet, anahtarları rastgele bir şekilde yuvalara eşlemlemelidir. Anahtarları gerçekten rastgele dağıtmalıdır. Buna ortalama durum varsayımı diyoruz.;**basit tekbiçimli kıyım fonksiyonu varsayımı**.. Bunun anlamı, S'nin

içindeki her k anahtarının, T'deki yuvalara kıyımlanma olasılığının eşit olduğudur. Bu noktada bağımsızlık varsayımını kullanarak, her eşlemlemenin diğer kayıtların, diğer anahtarların nereye kıyımlandığından bağımsız olacağını da belirtmeliyiz. Bu nedenle bir bağımsızlık koşulu da içeren bir varsayım da yapacağız. Yani iki anahtarın aynı yere kıyımlanmasının ihtimali nedir?

Bu varsayım çerçevesinde, m adet yuvam varsa iki anahtarın aynı yuvaya kıyımlanmasının olasılığı nedir acaba? Bir bölü m. Bir anahtarın onbeşinci yuvaya kıyımlanmasının olasılığı nedir? Bir bölü m. Anahtarlar dağıtılıyor ama iki anahtarın aynı yuvaya kıyımlanmasının olasılığı 1/ m'dir. Şimdi tanımı yapalım. Sorunuz mu vardı? Hayır. Pekiyi öyleyse. n adet anahtarı m sayıda yuvaya yönlendiren bir kıyım tablosunun loadfactor yada **yük oranı**α'dır, yani n bölü m dir. Düşünecek olursanız bu aynı zamanda yuva başına ortalama anahtar sayısıdır.

Yani α , ortalama anahtar sayısı bölü tablonun yük oranıdır. Tamam mı? Ortalama kaç anahtarım var? Öncelikle beklenen başarısız arama süresine bir bakalım. Başarısız arama deyince aslında tabloda olmayan bir şeyi arıyorum anlamına gelir. Sonuç nil olacaktır. Tabloda olmayan bir anahtarı aradığımda arama süresi ne çıkar? Θ Bir düzeyinde çıkacaktır; Öncelikle kıyım fonksiyonunu hesaplamak gibi belirli işler yapacağım. Bu durumda düzey en azından bir, artı listede arama yapmak zorundayım ve ortalamada listenin ne kadarında arama yaparım?

Bu listede arama yapmanın maliyeti nedir? Ortalamada. Eğer rastgele arama yapıyorsam. Tabloda olmayan bir anahtarı arıyorsam. Hangi iş olursa olsun, listenin sonuna kadar gitmek zorundayım, değil mi? Bu durumda tablodaki tüm yuvaları aramanın ortalama maliyeti nedir? Alfa. Doğru değil mi? Alfa. Listenin ortalama uzunluğu budur. Böylece kıyımlama yapmanın ve yuvaya erişimin maliyeti ve şu da sadece listede arama yapmanın maliyetidir.

Böylece beklenen başarısız arama süresi temelde alfa ile orantılıdır ve alfa birden büyükse α düzeyindedir. α birden küçükse, bir sabittir. Pekiyi, beklenen arama süresi ne zaman $\Theta(1)$'e eşittir? Bu ne zaman 1 düzeyindedir? Aslında basit sorular soruyorum. Ben sadece basit sorular sorarım. Bazıları zor sorular da soruyorlar. Evet. O noktaya iki adımda geleceğiz. Alfa bağlamında, ne zaman?

α sabit olduğu zaman. Alfa sabit olmak zorunda değil. Sabitten küçük de olabilir. Eğer α özellikle O(1) olursa, değil mi? Ya da sizin söylediğiniz gibi eğer n = O(m) ise.. Başka bir deyişle tablodaki elemanların sayısı olan (n) , m düzeyi tarafından, yani bir sabitle m'nin çarpımı ile yukarıdan sınırlanmışsa arama maliyeti sabittir. Birçok kişi size kıyım tablosunun sabit arama süresinde çalıştığını söyleyecektir. Bu aslında yanlıştır. Bu durum kıyım tablosunun yük oranına bağlıdır. Bazıları kıyım tablolarını

bu açıdan yanlış yorumlayarak programlama hataları yapmışlardır. Çünkü ellerindeki kıyım tabloları yerleştirecekleri eleman sayısına göre küçüktür. Bunun yararı olmaz.

Sayı giderek büyüyecektir; bu bir artı m bölü n olduğundan, büyüme n'ye oranlı olacaktır.Yani m'nin n ile başbaşa gitmesini sağlayamazsanız, bu bir sabit kalmaz. Şimdi başarılı bir arama olması için bir artı alfa olması gerektiğini ortaya çıkarır. Bunun için de birazcık matematikle uğraşmanız gerekir çünkü bu durumda aramayı tablodaki öğelerin aramasına uyarlamanız gerekiyor. Ama bu da bir artı alfa çıkacaktır ve bununla ilgili bilgiyi kitapta bulabilirsiniz. Bununla ilgili resmi kanıtlamayı da.. Burada beklenti yadabeklenen konusunu biraz yüzeysel geçtim ve probleme sezgisel yaklaştım. Bu nedenle bu iki konuyu da kitaptan okumalısınız.
Bu anlattıklarım, kıyım fonksiyonunun popular bir metot olmasının nedenlerinden sadece birisidir. Oluşturduğunuz tablo, o tabloya yerleştireceğiniz elemanların sayısından çok küçük değilse, bu yöntem dinamik bir kümeyi işlem başına O(1) maliyetiyle tanımlamanızı sağlar; araya yerleştirme, silme ve benzeri işlemleri, işlem başına sabit maliyetle yaparsınız. Sonra da tüm işlemler sabit zaman alır. Ama bu basit tekbiçimli kıyım fonksiyonu varsayımına sıkıca bağlıdır. Bu nedenle hangi kıyım fonksiyonunu seçerseniz seçin, ben yine de o fonksiyonla kötü sonuç alacağınız elemanlar bulabilirim. Bunlardan bir takım üretirim, fonksiyonun onları nereye götürdüğüne bakıp aynı yere gidecek başka elemanlar grubu üretebilirim.

Bunu engelleyebilecek bir yolu göreceğiz ama pratikte insanların anlayışı, öğeleri kullanmak zorunda olan çoğu programın gerçekte kıyım fonksiyonunda tersine mühendislik yapmadığı yönündedir. Bu nedenle pratikte iyi çalışan çok basit kıyım fonksiyonları vardır. Böylece bir kıyım fonksiyonu seçerken, anahtarları yuvalara tekbiçimli olarak dağıtmasını arzularız -- ve ayrıca anahtar dağıtımındaki bu düzenliliğin tekbiçimliliği de etkilememesini isteriz.

Örneğin sıkça göreceğiniz bir düzenlilik listeye sokulan bütün anahtarların çift sayı olmasıdır. Verinin bu özelliği vardır ve sadece çift sayıları kullanırız. Pratikte bir çok makine bayt işaretçileri kullanır ve bunlarda sıralama yaparken, örneğin dizilimlerin dizinlerini yada benzerlerini sıralarken kullanılan sayılar dörde bölünebilir niteliktedir.

Veya sekize bölünür niteliktedir. Onun için anahtar dağıtımındaki düzenliliğin yuva dağıtımını etkilemesini istemezsiniz. Bunu sağlamak için çabuk bir kıyım fonksiyonunda kullanılan en yaygın yöntem bölme metodu denilen bir yöntemdir. Burada Yaklaşım, kıyım fonksiyonu h(k)yı k mod m seçmektir; burada **m** tablodaki yuvaların sayısıdır. Bu pratikte oldukça iyi çalışır ama büyüklük yani mod seçiminde dikkatli olmanız gerekir.

Başka bir ifadeyle seçebileceğiniz her tablo boyutunda iyi çalışmayabilir. Şansınıza, kıyım tablolarını oluştururken tablonun kesin boyutuyla ilgilenmezsiniz. Tabloyu belli bir boyutta seçerseniz çoğunlukla bu uygun olacaktır, çünkü başarımı etkilemeyecektir. Bu nedenle kesin bir değer seçmek gerekmez. Özellikle **m**'yi küçük

bir bölenle seçmeyin; bu kıyım fonksiyonu için böylesi bir seçimin neden kötü bir fikir olduğunu size göstereceğim.

Aslında**d** küçük böleni demeliydim. Örneğin, d 2 ise, başka bir deyişle m çift sayıysa, yani biraz önce söylediğim gibi aynı zamanda tüm anahtarlar çift sayıysa, kıyım tablomun kullanımına ne olacaktır? Yani elimde çift sayılı yuvalar var ve kıyım tablomun kullanıcısının seçtiği anahtarların hepsi de çift sayılı; bu durumda kıyım tablomun kullanımı açısından ne olacak dersiniz? En kötü durumda hangi kıyım fonksiyonunu seçersem seçeyim hepsi aynı yuvayı işaret edecektir.

Burada belirtmemiz gereken kıyım fonksiyonumun aslında iyi iş yaptığı ama bu özellikten etkilendiğidir. Hangi anahtar kümesini seçersem seçeyim bu durumu yaratan özellik acaba nedir? Kıyım tablosuna ne olacaktır? Elimde çift sayılar var,bu mod çift sayı. Bu kıyım fonksiyonu açısından ne anlama gelir? Çift sayıdır, değil mi? Çift sayı olan bir mod var. Tablomun kullanımında ne olacak acaba? Evet, hiçbir zaman tek sayılı yuvalara kıyım yapamayacaksınız. Yuvaların yarısını boşa harcadınız. Anahtar dağıtımının ne olduğu hiç önemli değil.

Bunlar çift olduğu sürece tek sayılı yuvalar hiç kullanılamayacak anlamı çıkıyor. Şimdi başka bir uç örnek vereceğim; m = 2^rolduğunu düşünün. Başka bir deyişle tüm faktörleri küçük bölenler olsun, tamam mı? Bu durumda, kmodm almayı düşündüğümde kıyım k'nın bütün bitlerine bağımlı bile olmayacak,öyle değil mi? Örneğin, elimde 1011000111011010 anahtarı varsa,r burada 6 dır ve m = 2^6olacaktır.

Bu ikili sayıya bakarsam, ve mod ikinin altıncı kuvveti alınırsa olursa, kıyım değeri ne olacak? Bir şeyin modunu ikinin bir kuvveti olarak alırsam fonksiyon ne olur? Şimdi bu fonksiyonu kıyımlıyorum. k ikili sistemde olsun. Ve ben ikinin altıncı kuvvetine göre mod alıyorum.Pekiyi modu iki kabul etseydim cevap nedir? Bu sayı mod 2 olursa ne olur? Bu sayı mod 2'de ne olacaktır 0. Bu sayı mod4 olursa ne çıkar? 10. Pekiyi, mod2^6 ise sayı ne olur? Evet sadece bu son altı bitten oluşur. Bu h(k)'dır. Yani bir şeyi mod ikinin bir kuvveti olarak seçerseniz,yaptığınız şey o sayının sadece düşük değerli bitlerini hesaba katıyorsunuz demektir. Örneğin mod ikinin r'ninci kuvvetiyse, sayının r adet düşük değerli bitini hesaba katarsınız. Yani kıyım fonksiyonu buradaki daha yüksek değerlere bağımlı bile değil.

Bu bayağı kötü bir durumdur çünkü veri yapılarında çoğunlukla görülen birdüzenlilik, düşük düzeydeki bitlerin aynı olması ve yüksek düzey bitlerinin farklı olmasıdır; yada bunun tam tersi olabilir.Bu örnekteki yapı iyi değildi. Bu

konuda iyi bir buluşsal yaklaşım,sağlamak gerekir; çünkü2 ve 10pratikte çokça görünen ortak tabanlardır. Bazen, asal sayılar da uygun olmayabilir. Ama asalları bulmak oldukça kolaydırve asallarla ilgili birçok güzel teorem vardır.Bu durumda, ne olduğunu bildiğiniz bir şeyin kodunu yazarken, asalları bir kitaptan, internetten bulabilir yada küçük bir kod yazıp bir asal sayı seçebilirsiniz.

İkinin ve onun kuvvetlerine çok yakın olmazsa, büyük olasılıkla iyi iş görecektir. İyi çalışacaktır. Yani bu bölme metodu çok yaygın bir metottur. Tamam,ancak şimdi göreceğimiz metot çoğu kez daha da üstün. İnsanların bunu kullanmasının nedeni kodlarının içine yazabilmesidir. Ama en iyi metot da değildirve bunun nedeni de bölme işlemidir, çünkü bir çok bilgisayarda bölme, çarpma veya toplamaya oranla daha fazla işlem gerektirir. Bu nedenle de bu metotta aslında birkaç çarpma işlemi uygulanır. Yani göreceğimiz metot genelde iyidir amabugün konuştuğumuz kıyım fonksiyonu metotlarının hiçbirisi iyilikleri kanıtlanmış metotlar değildir.

Şimdi, çarpma metodunun iyi yanı sadece çarpma işlemine ihtiyaçduyduğudur. Bu nedenle çoğunlukla uygun olacağı için yuvaların sayısını ikinin kuvvetleri olarak varsayabileceğiz. Ve bu işlemde bilgisayarın w-bitlik sözcükler kullandığını varsayacağız. Bilirsiniz 32 yada 64 bitlik sözcükler bilgisayarlar için uygun. Böylece kıyım fonksiyonu,h (k)= A çarpı k mod 2 üzeri w sağa kaydırılmış w-r olacaktır.

Buradaki işlemin anahtar bölümü A' bir tek tamsayıdır --ve 2^{w-1} ile 2^{w} arasında seçilmiştir.. Bu tek tamsayı bilgisayar sözcüğünün tüm uzunluğudur. Yapacağınız şey, bunu anahtarınız neyse onunla çarpmaktır; yani buradaki komik tamsayıyla.. Sonra onu 2^{w}moda taşırsınız. Sonra da sonucunu alıp şu sabit değerde, (w-r) kadar sağa kaydırırsınız. Bu bit bazında bir sağa kaydırma işlemidir.

Şimdi bunun ne yaptığına bir bakalım. Ama önce size A'nın seçiminde yapmanız ve kaçınmanız gereken şeyler konusunda birkaç ipucu vereyim. A'yı, ikinin bir kuvvetine çok yakın seçmemelisiniz. Bu metot oldukça hızlıdır çünkü 2^{w}modululoluçarpma bölmeden daha hızlıdır. Ardından yapılan sağa kaydırma da hızlıdır çünkü önceden bildiğiniz bir değerle çalışırsınız. Bu değeri kıyım fonksiyonuna başlamadan bilirsiniz çünkü w ve r önceden bildiğiniz değerlerdir.

Burada derleyici sıkça bazı aldatmalar yaparak işlemi daha da hızlandırır. Şimdi bir örnekle bu kıyım fonksiyonunun nasıl çalıştığını görelim. Yuva sayısı 8, yani ikinin üçüncü kuvveti. Ve elimizde yedi bitlik garip bir sözcük boyutu olsun. Kimse yedi bitle çalışan bir bilgisayar biliyor mu? Bilmiyorsanız bile işte burada bir tane var. Böylece bütün anahtarlarımızın kıyımında kullanılacak sabit değerimiz A'dır. Bu örnekte bunu 1011001 alalım. A budur. Şimdi çarpma işleminin diğer öğesi olan k'ya bir değer vereceğim. k= 1101011 olsun. Bu da benim k değerim.

Bunları çarparım. Çarpmadan önce herbiri tam birsözcük uzunluğundadır. Bunları makinenin tam sözcük uzunluğu olarak görebilirsiniz; bu durumda 7 bit. Pratikte ise bu 32 bit uzunluğunda olur ve anahtarımı da düşünürsek iki 32 bitlik sayıyı çarpıyorum. Tamam, bu çarpmayı yaptığımda 2w bitlik bir cevap bulurum. Yani iki adet w bitlik sayıyı çarptığınızda 2wbitlik bir cevap çıkar.

Örneğimizde şu sayı çıkıyor: 10010100110011, tamam mı? Bu çarpımın sonucudur. Sonra bunu mod2^walacağız. 2^wmodulonun amacı, bu çarpımdaki yüksek düzeyli bitleri görmezden gelmektir. Böylece bütün bunlar ihmal edilecektir; hatırlayacak olursanız bir şeyi mod2'nin bir kuvvetine taşıdığımda sadece düşük değerli bitler kalıyordu. Bu nedenle modülo olarak buradaki düşük değerli bitler kalacaktır. Sonra da sağa kaydırma işlemini yaparım; bu da iyidir, çünkü birçok makineye iki 32 bitlik sayı çarpıldığında, düşük düzeyli 32 biti çıkışa vermesi için talimat yüklenmiştir. Ve bu talimat sayesinde işlem 64 bitlik tam yanıtı verme durumundan daha hızlı yürütülür. Yani bu çok uygundur. İkinci bir husus da, ben sadece sözcüğün bu örnekte yüksek düzeyli üç bitini alıyorum.

Böylece bu benim h(k) değerim oluyor. Bunlar da sözcüğü sağa kaydırdığım için yok oluyorlar. Yani şunu sağa kaydırdığımızda daha yüksek düzeyli bitteki sıfırlar geliyor ve h(k)'nın değerini elde ediyorsunuz. Tamam, burada neler olduğunu, bunun neden iyi bir metotolduğunu ve neleri sağladığını daha iyi anlamak için yaklaşımlardan birisi, A'yı ikili kesirli bir sayı olarak hayal etmektir.

Ondalık noktasının, af edersiniz ikili noktanın yani yada taban yani radix noktasının yerinin burası olduğunu hayal edin. Birşeyleri çarptığımda ikili nokta şuraya gelir. Kavramsal olarak düşündüğünüzde, bunu donanıma vermem gerekmez çünkü biz zaten donanımın yaptığını yapıyoruz. Ama ben bunun orada da burada da ne olduğunu hayal edebilirim. Sonunda A'yı bir sayının

kesiri olarak alırsam, bu çarpımın kesirli kısmıdır. Yani biz bu işlemi modüler bir tekerlek yada modülolu bir rulet gibi görebiliriz.

Şimdi burada bir tekerlek var ve ben bunu sekize bölüyorum; bu nokta sıfır. Sonra etrafında döneceğim ve burası bir olacak. Dönmeye devam edeceğim, bu nokta iki olacak ve bu böyle sürecek; tüm tamsayıları sıfır noktasının etrafına dizili şekilde, bu birim tekerleğinin çevresince sıralayacağım.
Sonra bunları kesirli parçalara bölebiliriz. Bu temelde sıfır noktasıdır. Bu 1/8'dir çünkü sekize bölüyoruz ve iki, üç, dört, beş, altı, yedi diye gider.

Böylece, bu durumda bir çarpı A dediğimde, çarpma işlemi sonucu bir kere A yaklaşık şuraya gidecektir çünkü çarpım yaklaşık beş buçuk olacaktır; yada aslında bölüm sekiz olacaktır.Bu beni yaklaşık şuraya götürür. Bu A'dır. Şimdi 2A 'yı uygularsam, çevrede dönme sürer ve beni yaklaşık nereye götürür? Şurayı biraz gececek şekilde buraya getirir. Bu 2A idi. Sonra $3A$ yaklaşık beni şuraya götürür. Böylece yeni bir A eklediğimde beni çevrede bir A mesafesinde dolaştırır.

Buradaki fikir şudur: Eğer A, örneğin tek sayı ise ve ikinin kuvvetlerinden birine çok yakın değilse, atamayı başka bir yerdeki farklı yuvaya yapar. Böylece etrafta dolaşırken k çok büyük bir değerse, k çarpı A çevrede k kere döner. Nerede son bulur? Bu şans tekerleği yada benzerini çevirmek gibidir. Tamam, bir yerlerde son bulur ve temeldeki kavram da budur. Temeldeki kavram bir yerde sona ereceğidir. Yani temelde aradığınız kA'nın nerede son bulacağıdır. Döngü sürer ama bir yerde durur. Duracağını bilmek de iyi bir şeydir. Ama bunlar kıyım fonksiyonuna yalnızca buluşsal yaklaşımlardır çünkü her kıyım fonksiyonu için kötü sonuç verecek anahtarlar bulabilirsiniz.

Soru pratikte ne kullanıldığıdır? Bunu ikinci konumla ilişkilendirmek istiyorum; şu ana kadar -- şu ana kadar çarpışmaları zincirleme yoluyla çözmeye çalıştık.Çarpışmaları çözme konusunda genelde yararlı olan başka bir yaklaşım daha vardır; çarpışmalar open addressing yada **açık adresleme** denen bir yöntemle dehalledilebilir. Bu metoddaki ana fikir linkler için depo kullanılmamasıdır. Linklemeyle çarpışmaları çözmeyi amaçladığımızda, bu iş için her kayıtla ilgili ek link alanına ihtiyaç vardı. Bu büyük bir gider olmayabilir ama bazı uygulamalarda bu kayıtlara dokunmak istemeyebilirim. Bu durumlarda açık adresleme çarpışma çözmede yararlı bir yol olabilir.

Açık adreslemedeki fikir, bir yuvaya kıyımlama yaptığımda o yuva dolu çıkarsa, yapmam gereken şey ikinci ve başka bir kıyım fonksiyonunu devreye

sokmaktır. O yuvayı kontrol ederim. Eğer yuva doluysa tekrar kıyımlama yaparım. Bu sonda dizisini boş bir yuva bulana dek sürdürürüm ve bu işlemin daha önce kontrol ettiğim şeyleri tekrar yapmamı engelleyen bir permütasyon olmasını umarım. Ve iyi bir sonda dizim olursa iyimser bir yaklaşımla boş bir yeri çabucak bulurum. Sonra aramak için de aynı sonda dizisini kullanırım.

Yani buradaki düşünce tabloyu sistematik olarak sondalayarak boş bir yuva bulmaktır? Bu düşünceyi kıyım fonksiyonu dizisinin iki bağımsız değişken alıp almayacağına bakarak genişletebiliriz: Bir anahtar ve bir de arama adımı. Başka bir deyişle bu sıfırıncıdır veya birincidir, ikincidir, gibi.Yani iki bağımsız değişkene gereksinim vardır. Böylece h,evrenimizdeki anahtarlarıve sonda numaramızı bir yuvaya gönderir.

Bu bizim anahtarlar evreni. Bu sonda numarası probenumberve bu da slot yani yuva. Şimdi daha önce de belirttiğim gibi sonda dizisi permütasyon olmalıdır. Diğer bir deyişle 0'dan m-1'e kadar olan sayıların rastgele sıralanmasından oluşmalıdır. Yalnızca yeniden düzenlenmeleri gerekmelidir. Açık adresleme bağlamındaki diğer bir husus da, tablo dolacak kuşkusu ile n ilmiklemesiyle ilgili endişe duymanıza gerek olmadığıdır.

Yani, tablodaki eleman sayısı n, tablonun boyutu olan m'den küçük eşit olmalıdır, çünkü tablodaki yuvalar dolabilir. Ve eğer dolarsa her yerde arama yapmak zorunda kalırsınız veelemanı koyacak bir yeri hiçbir zaman bulamazsınız.bulunmaz. Ve son olarak da bu tür veri tanımlamada silme işlemi yapmak zordur. İmkansız değildir; silme işlemi yapabilecek veri tanımlama uygulamaları vardır. Ama temelde bu zordur, çünkü siz bir anahtarı tablodan çıkardıktan sonra, bir sonda dizisiyle elemanını arayan birisinin yuvayı boş bulma tehlikesi vardır.Bulamayınca da "aradığım anahtar herhalde burada değil",.der. Dolayısıyla bu konuyla ilgilenmek zorundasınız. Bir şeyleri silebilirsiniz ama bunları imleyip korumanız gerekir ve bu amaca hizmet eden birçok veri tanımlama yöntemi vardır.

Ama bu zor bir iştir. Zincirlemeye oranla karmaşıktır; zincirlemede elemanı zincirden sadece çıkarırsınız. Bir örnek yapalım ve aynı sayfada olduğumuzdan emin olalım. Şimdi bir anahtarı tabloya sokacağız. k= 496 olsun. Bu da benim tablom ve içinde bazı değerler var: 586, 133, 204, 481, gibi. Tablo böyle görünüyor; diğer yerler boş. Sıfırıncı adımımda h(496,0)'ı sondalayacağım. Bu beni diyelim ki 204 yuvasına götürdü. Ve ben derim ki: "Burada bir şey var."Burası dolu.

Yeniden sonda yapmam gerekir. Ben de h(496,1)'i sondalarım. Belki bu beni şuraya eşlemler ve orada bir şey olduğunu görüyorum. Sonra h(496,2) sondasını yaparım. Belki bu beni şuraya götürür. Burası boş. Eğer arama yapıyorsam nili raporlarım. Eğer tabloya sokmak istiyorsam oraya yerleştiririm. Eğer o değeri arıyorsam, oraya yerleştirdikten sonraki aramada aynı sırayı takip ederim. Bu şeylerin meşgul olduğunu belirlerim ama sonunda aradığım değere ulaşır ve onu keşfederim.

Bunun yanında, insanların kullandığı çeşitli buluşsal yöntemler de vardır; örneğin bir araya yerleştirme yapmak için global boyutta yapılacak en büyük sonda sayısından daha fazla sonda yapmakta bir yarar olmadığından, en uzun sonda dizisinin kaydı tutulur. Yani, 5 kez yapmışsam, 5, araya yerleştirme için yapmış olduğum en fazla sonda sayısıysa,bir arama hiçbir zaman beşten fazla yapılmamalıdır, ve kıyım tabloları bazen bu ek bilgiyi hatırlayarak bir şey bulamadan devam etmek yerine aramadan vazgeçerler.

Evet, aramada da aynı sonda dizisini kullanır. Başarılı olursa kaydı bulur; başarısız olursa nil yani boş değeri elde edilir.Yani oldukça doğrudur.Bir kez daha tekrarlarsak, aynı kıyım fonksiyonlarıyla başlandığı gibi, bir sonda dizisinin nasıl oluşturulacağı, bu işin nasıl verimli kılınacağı konusunda pek çok fikir vardır. Bunların en basitine linearprobing, yani **doğrusal sondalama** denir. Bu durumda h(k, i) =. [h(k,0) + i]mod m değeri olur.

Af edersiniz, orada h üssü yerine sadece h olacak. Burada olan biten, yani buradaki fikir, yapacağınız tek şeysıfırıncı sondada getiriph(k,0)'a bakmaktır. Sonra, sonda birde sıradaki yuvaya bakarsınız. Sonra ikide bir sonraki yuvaya bakarsınız. Yani, sıralamada oradan buraya atlamak yerine basitçe sondalama yaparak sıradaki en uygun yuvayı bulursunuz. Böylece, yapmanız gereken mod m boyunca aşağıya doğru taramaktır. En alt düzeye geldiğinizde tekrar en üste çıkarsınız. i'ninci aramayı gerçekleştirmek kolaydır çünkü her defasında tüm kıyımlama fonksiyonunun yeniden hesaplamak zorunda değilsiniz. Bütün yapmanız gereken her defada 1 eklemek, çünkü bununla bir önceki arasındaki fark sadece birdir. Sadece aşağıya ineceksiniz. Buradaki sorun, bir gruplandırma olgusuyla karşılaşacak olmanızdır. Belirli bir alana bir şeyler yığdığınızda, herkes bunları sonuna kadar aramayı sürdürmek zorundadır.

Böylece bunun iyi veri tanımlama yöntemlerinden biri olmadığı ortaya çıkar ama hızlı ve biraz da kural dışı bir işlem yapmak istediğinizde o kadar da kötü değildir. Bu uygulama, kıyım tablosunun çok dolu olan bölümlerinde

primaryclustering yani birincil gruplandırma nedeniyle sıkıntı yaratmaktadır. Bu nedenle o bölgeye kıyımı yapılacak veri için oradaki her şeye bakılması zorunluluğu vardır. Bunun sonucu da dolu yuvalarınuzun dizilerinin varlığıdır. Uygulamalarda kullanılan karesel gruplandırma benzeri yöntemler de vardır ve bunlarda her seferinde 1 eklemek yerine **i** eklersiniz. En çok kullanılan veri tanımlama yöntemi, **çift kıyımlama** adı verilen bir yöntemdir; bunun iyi bir veri tanımlama yöntemi olduğunu göstermek için istatistikler, çalışmalar yapılmıştır.

Bunu, h(k,i) ile başlayacak eşitliği aşağıda yazayım çünkü burada yerim var. h(k,i) =mod m işlemi altında [h_1(k) + i h_2(k)]. Böylece mod m üzerinde iki kıyım fonksiyonunuz olur. h_1(k) ve h_2(k) şeklinde iki kıyım fonksiyonunuz var. Bu iki kıyım fonksiyonunu hesaplamak için önce h_1(k)'yı sıfır sondasında kullanırsınız, çünkü burada i sıfır olacaktır. Tamam mı? Sonra bir numaralı sonda olarak h_2(k)'yı eklersiniz.

İki numaralı sonda olarak bu kıyım fonksiyonu yani h_2 (k)'nın değerini tekrar eklersiniz. Ardından yapacağınız her sonda için h_2(k)'yı eklemeyi sürdürürsünüz. Yani oldukça kolaydır; önceden iki kıyım fonksiyonunu hesaplarsınız yada ikincisini dilerseniz sonraya bırakırsınız. Ama özünde ikisi de önceden hesaplanır ve ikincisi devamlı olarak eklenir. Birincisinin olduğu yerden başlarsınız ve sonda dizilerinizi belirlemek için, modm de ikincisini eklemeyi sürdürürsünüz.

Bu aslında mükemmel bir metottur. İyi iş çıkarır ve burada m'yi genellikle ikinin bir kuvveti olarak seçersiniz, çünkü bu çoğunlukla çarpma metoduyla kullanılır. Örneğin, m ikinin bir kuvveti olduğunda h_2(k)'yı tek sayı olmaya zorlarsınız. Bunda çift değerli sayı kullanmayız çünkü bunu yaparsak belirli bir anahtarı atlama durumu ortaya çıkabilir. Öte yandan ilerlerken her şeyin çift olması yada her şeyin tek olması sorunuyla da karşılaşabilirsiniz. Ama h_2(k)' yı tek sayı, m'yi de ikinin kuvveti yaparsanız, her yuvaya erişmeniz garantili olur. Şimdi bu veri tanımlamasını çözümleyelim. Bunu çözümlemek aslında bayağı ilginç olacak.

Bu işlemde bayağı hoş matematik var. Şimdi tekrar edeyim, en kötü durumda kıyımlama iyi değildir. Bu nedenle biz ortalama durumu çözümleyeceğiz. Bunun için de zincirleme yönteminde olandan daha kuvvetli bir varsayıma ihtiyaç olacak. Ve biz buna **tek biçimli kıyımlama varsayımı** diyoruz; bu varsayım, "her anahtar, kendi sonda dizisinin m faktöryel permütasyonları içerisinde eşit olarak, ve diğer anahtarlardan bağımsız olarakyer alır." şeklinde ifade edilmektedir.

Ve burada kanıtlayacağımız teorem, beklenen arama sayısının en çok, alfanın birden küçük olması durumunda, bir bölü bir eksi alfa olduğudur, tamam mı? Yani tablodaki anahtar sayısının yuva sayısından daha az olduğu durumdur. Kanıtlayacağımız şey arama sayısının bir bölü bir eksi alfa olduğudur. Burada alfa yük oranıdır ve tabii açık adreslemede yük oranının birden az olmasını isteriz, çünkü yuvalardan daha fazla anahtarlarımız olursa açık adresleme çalışmaz. Çünkü tabloda her anahtar için bir yuva bulmak zorundasınız. Bunu kanıtlamak için başarısız bir aramaya bakacağız.

Öncelikle, bir sonda her zaman gerekli olacaktır. Bu durumda elimde n/m olduğunda, af edersiniz, n öğenin m yuvada depolanmış olduğu durumda bir sondayaptığımda, tabloda olan başka bir şeyle çarpışma olmasının olasılığı nedir? Bir çarpışmaya neden olmamın olasılığı nedir? Evet? Evet,n/m, öyle değil mi? Yani n/m olasılıkla bir çarpışma olacaktır, çünkü tablomda n adet öğe var.

Bunlardan birine rastgele kıyımlama yapıyorum. Bu durumda bir şeye çarpmamın olasılığı nedir? n/m. Sonra ikinci bir sonda gerekli olacak. Öyleyse ikinci sondalamayı da yaparım. Pekiyi ikinci sondalamamda bir çarpışma olmasının olasılığı nedir? Burada tekbiçimli kıyımlama varsayımını yapacağız. Her anahtar, kendi sonda dizisinin m faktöryel permütasyonlarında eşit yer alır. Bu koşulda ikinci sonda işleminde bir çarpışma olması olasılığı nedir?

Evet? Bu bir permütasyonsa öyle olmaz değil mi? Ona benzer bir şey. Tam olarak nedir? Soru bu. Tamam, bu bir permütasyon olduğundan aynı yuvaya düşmezsiniz. Evet? Bu kesinlikle doğru. (n -1) /(m -1), çünkü ilk düştüğüm yuvayı devre dışı bıraktım.İlk sondadaki anahtar bir yuvaya yerleşmişti.Şimdi kalan n eksi bir yuvaya rastgele bakıyorum ve bu yuvalarda topluca m eksi bir anahtar var.

Herkes bunu kavradı mı? Bu olasılıkla bir çarpışma elde ediyorum. Bunun anlamı üçüncü sondalamaya ihtiyacım olduğudur, öyle değil mi? Ve böylece devam ederiz. Pekiyi bir sonraki olasılık ne olur? Evet, n eksi iki bölü m eksi iki olur. Şunu not edelim. Öyleyse şu n – i / m – i, n / m'den daha küçüktürbuda alfaya eşittir. Tamam mı? Yani, n eksi i bölü m eksi i, n bölü m'den yani alfadan daha küçüktür.

Bu bağlamda mantık yürütürsek, eğer n, m'den küçükse, her i çıkarması işleminde n'nin m'ye oranla daha büyük bir kısmını azaltıyorum demektir.Bu

nedenle n eksi i bölü m eksi i, n bölü m'den daha küçük olacaktır. Şimdi cebir işlemlerini yapabilirsiniz; cebir kullanmak, nitelik açısından neler olduğunu anlasanız bile nicelik açısından yardımcı olur.

Böylece beklenen arama sayısı şuna eşit olacaktır: Burada biraz yere ihtiyaç var; zorunlu olarak önce 1 var çünkü bir sondalama yapmak zorundayız, artı bunun olasılığı n /m. Bir arama işlemi daha yapmam lazım olduğundan 1 artı bunun olasılığı n -1 / m - 1. Sonraki sondalama işleminden1 artı n -2 / m -2 ve bu işlemi 1 artı 1 / m - n'ye varana kadar sürdürmem gerekli. Yani her işlem öncekilerle peş peşe bağlanıyor. Kitapta göstergesel rastgele değişkenler yoluyla bunun daha kesin kanıtlanmasını bulabilirsiniz. Ben size daha kısa yolunu göstereceğim. Şimdi, bu temelde benim birinci sondalamam; n bölü m olasılığıyla. İkinci sondalamayı yapmak zorundayım. Ve bunun sonucunda olasılık n eksi bir bölü m eksi bir çıkınca, bir tane daha yapmak zorundayım. Ve bunun da sonucu n eksi iki bölü m eksi iki çıkınca, bir tane daha yapmak zorundayım ve bu böyle devam ediyor.
Böylece yapmam gereken sonda sayısı ortaya çıkıyor: Bu bir artı alfa,çarpı bir artı alfa, çarpı bir artı alfa,… çarpı bir artı alfa'yaeşit yada küçüktür değil mi?Burada olan gerçek çerçevesindetabii ki .Ve bu da neye eşit yada küçük onu bulmak için çarparım bunları. Alfa artı, alfa kare artı, alfa küp artı, ve böylece gidiyor.Bunu sonsuza kadar götürebilirim Bunu sınırlayacak bir değer arıyorum.

Herkes buradaki matematiği anlıyor mu? Bu bir toplam, i sıfırdan sonsuza giderken alfanın i nci kuvvetlerinin toplamı; bu toplam da 1 bölü bir eksi alfaya eşittir; bu da bildiğiniz geometrik dizi toplamıyla oluşturulur. Kitapta başarılı aramanın çözümlenmesi de var ve bu biraz daha teknik düzeyde çünkü tabloda zaten olan bir değeri aradığınızda tablodaki dağılımı da hesaba katmak zorunda olursunuz. Ama sonuçta oradaki sınır da bir bölü bir eksi alfa çıkar.

Şimdi bunun ne anlama geldiğine bir bakalım.Eğer alfa birden küçük bir sabitse bu büyük O (1)düzeyinde sondalamanın gerekeceğini gösterir. Yani alfa sabitse birinci düzeyde arama işlemi gerekir. Ama bu sabite ne olduğunu da anlamakta yarar var. Örneğin tablonun % 50'si doluysa,yani alfa yarıma eşitse, bu çözümlemede beklenen sonda sayısı nedir? Yapılan bu analizle?İkidir, çünkü bir bölü bir eksi yarım ikiye eşittir.

Eğer tablomun % 90'ının dolmasına izin verirsem, ortalamada kaç sonda işlemi yaparım? Ortalama. On. Gördüğünüz gibi tabloyu doldurdukça

maliyetciddi biçimde artıyor. Bu nedenle genelde tablonun çok dolmasına izin vermezsiniz. Yani 99,9 yüzde kullanımı istememelisiniz. Elimde tümü dolu olan harika bir kıyım tablosu var demek, yavaş bir uygulama var demekle aynı anlama gelir. Çok çok yavaş olacaktır, çünkü alfa bire yaklaştıkça aslında m veya n'ye yaklaşacaktır zaman..

Tamam. Gelecek sefere bana göre algoritmalardaki en ilginç fikirlerden birine bodoslama dalacağız. Hangi kıyım fonksiyonunu seçerseniz seçin kötü bir anahtar kümesi olması probleminin çözümünü konuşacağız. Yani gelecek derste bu problemin çok akıllıca çözümleri olduğunu göreceğiz. Ve bu amaçla da yoğun şekilde matematik kullanacağız; onun için çok eğlenceli bir derse hazır olun.

Dersimiz bitmiştir.