1. Veri Yapıları ve Algoritmalar

Konya merkezine en fazla 300 km uzaklıkta bulunan 50.000'den fazla insanın yaşadığı kaç şehir var? Şirkette çalışan kaç kişi yılda 100.000 liradan fazla kazanıyor? Bir bölgedeki tüm bilgisayarları 1.000 km kabloyla bağlayabilir miyiz? Bunun gibi soruları cevaplamak için gerekli bilgilere sahip olmak yeterli değildir. Bu bilgileri, ihtiyaçlarımızı karşılamak için cevapları zamanında bulmamızı sağlayacak şekilde düzenlemeliyiz.

Bilgiyi temsil etmek bilgisayar biliminin temelidir. Çoğu bilgisayar programının birincil amacı hesaplama yapmak değil, mümkün olduğunca hızlı bir şekilde bilgiyi depolamak ve almaktır. Bu nedenle, veri yapılarının ve onları kullanan algoritmaların incelenmesi bilgisayar biliminin merkezinde yer alır. Ve bu dersin konusu da budur - işlemeyi verimli bir şekilde yapmak için bilgiyi nasıl yapılandıracağınızı göreceksiniz.

Bu dersin üç temel amacı vardır. Birincisi, yaygın olarak kullanılan veri yapılarını sunmaktır. Bunlar, bir programcının temel veri yapısı araç takımını (toolkit) oluşturur. Birçok sorun için araç setindeki bazı veri yapıları iyi bir çözüm sunar.

İkinci hedef, ödünleşim (tradeoff) durumunu göstermek ve her veri yapısıyla ilişkili maliyet ve faydaların olduğu göstermektir. Bu, her bir veri yapısında tipik işlemler için gereken alan ve zaman miktarını tanımlayarak yapılır.

Üçüncü amaç, bir veri yapısının veya algoritmanın etkinliğinin nasıl ölçüleceğini öğretmektir. Araç setinizdeki hangi veri yapısının yeni bir problem için en uygun olduğunu ancak böyle bir ölçümle belirleyebilirsiniz. Sunulan teknikler, icat edilebilecek yeni veri yapılarını değerlendirmenize de olanak tanır.

Bir sorunu çözmek için genellikle birçok yaklaşım vardır. Aralarında nasıl seçim yaparız? Bilgisayar programı tasarımının merkezinde genellikle iki hedef vardır:

Anlaşılması, kodlanması ve hata ayıklanması kolay bir algoritma tasarlamak.
Bilgisayar kaynaklarını verimli kullanan bir algoritma tasarlamak.

İdeal olarak, ortaya çıkan program bu hedeflerin her ikisi için de doğrudur. Böyle bir programın "mükemmel" olduğunu söyleyebiliriz. Burada sunulan algoritmalar ve program kodu örnekleri bu anlamda zarif olmaya çalışsa da, bu dersin amacı, hedef (1) ile ilgili sorunları ele almak değildir. Bunlar

öncelikle Yazılım Mühendisliği disiplininin konularıdır. Bu ders çoğunlukla hedef (2) ile ilgili konular hakkındadır.

Verimliliği nasıl ölçeriz? Bir algoritmanın veya bilgisayar programının verimliliğini değerlendirmek için asimptotik analiz adı verilen bir yöntem kullanılır. Asimptotik analiz aynı zamanda bir problemin doğal zorluğunu ölçmenize de olanak tanır. Sunulan bir algoritmanın zaman maliyetini tahmin etmek için asimptotik analiz teknikleri kullanılır. Bu, verimliliği açısından aynı sorunu çözmek için önerilen her bir algoritmanın diğer algoritmalarla nasıl karşılaştırıldığını görmenizi de sağlar.

Veri Yapıları Felsefesi

Veri Yapılarına Neden İhtiyaç Var?

Her zamankinden daha güçlü bilgisayarlarla program verimliliğinin daha az önemli hale geldiğini düşünebilirsiniz. Sonuçta, işlemci hızı ve bellek boyutu hala gelişmeye devam ediyor. Bugün sahip olabileceğimiz herhangi bir verimlilik sorunu yarının donanımıyla çözülmeyecek mi?

Daha güçlü bilgisayarlar geliştirirken, bugüne kadar, daha karmaşık kullanıcı arayüzleri, daha büyük problem boyutları veya daha önce hesaplama açısından mümkün olmadığı düşünülen yeni problemler gibi daha karmaşık problemlerin üstesinden gelmeye çalışılmıştır. Daha karmaşık problemler daha fazla hesaplama gerektirir ve bu da verimli programlara olan ihtiyacı daha da artırır. Görevler daha karmaşık hale geldikçe, günlük deneyimlerimize daha az benzerler. Günümüzün bilgisayar bilimcileri, verimli program tasarımının ardındaki ilkeleri tam olarak anlamak için eğitilmelidir, çünkü bilgisayar programları tasarlarken onların sıradan yaşam deneyimleri çoğu zaman yeterli değildir.

En genel anlamda bir veri yapısı, herhangi bir veri gösterimi ve bununla ilişkili işlemlerdir. Bilgisayarda saklanan bir tamsayı veya kayan noktalı sayı bile basit bir veri yapısı olarak görüntülenebilir. Daha yaygın olarak, "veri yapısı" terimi veri öğeleri koleksiyonu için bir organizasyon veya yapılanma anlamında kullanırlar. Bir dizide saklanan sıralı bir tamsayı listesi, böyle bir yapılandırmaya örnektir.

Bir veri öğeleri koleksiyonunu depolamak için yeterli alan verildiğinde, koleksiyon içinde belirli öğeleri aramak, veri öğelerini istenen herhangi bir sırada yazdırmak veya başka şekilde işlemek veya herhangi bir belirli veri öğesinin değerini değiştirmek her zaman mümkündür. Böylece herhangi bir veri yapısı üzerinde gerekli tüm işlemleri yapmak mümkündür. Bununla birlikte, uygun veri yapısını

kullanmak, birkaç saniye içinde çalışan bir program ile birçok gün boyunca çalışan bir program arasında fark oluşturabilir.

Bir çözüm eğer problemi gerekli kaynak kısıtlamaları içinde çözüyorsa verimli kabul edilir. Kaynak kısıtlamalarına örnek olarak, verileri depolamak için mevcut toplam alanı (muhtemelen ayrı ana bellek ve disk alanı kısıtlamalarına bölünmüş) ve her bir alt görevi gerçekleştirmek için izin verilen süreyi içerir. Belirli gereksinimleri karşılayıp karşılamadığına bakılmaksızın, bir çözümün bilinen alternatiflerden daha az kaynak gerektirmesi durumunda bazen verimli olduğu söylenir. Bir çözümün maliyeti, çözümün tükettiği kaynak miktarıdır. Çoğu zaman maliyet, çözümün diğer kaynak kısıtlamalarını karşıladığı varsayımıyla, zaman gibi bir anahtar kaynak açısından ölçülür.

İnsanların sorunları çözmek için programlar yazar. Ancak, belirli bir sorunu çözmek için ulaşılması gereken performans hedeflerini belirlemek ve problemi analiz ederek, doğru veri yapısını seçmek gereklidir. Zayıf program tasarımcıları bu analiz adımını görmezden gelirler ve aşina oldukları ancak soruna uygun olmayan bir veri yapısı uygularlar. Sonuç tipik olarak yavaş bir programdır. Tersine, daha basit bir tasarım kullanılarak uygulandığında performans hedeflerini karşılayabilecek bir programı "iyileştirmek" amacıyla daha karmaşık bir yapıyı benimsemenin de bir anlamı yoktur.

Bir problemi çözmek için bir veri yapısı seçerken şu adımlar izlenir:

1) Desteklenmesi gereken temel işlemleri belirlemek için problem analiz edilir. Temel işlemler: veri yapısına bir veri öğesinin eklenmesi, veri yapısından bir veri öğesinin silinmesi ve belirli bir veri öğesinin bulunması olabilir.
2) Her işlem için kaynak kısıtlamaları ölçülür.
3) Bu gereksinimleri en iyi karşılayan veri yapısı seçilir.

Bir veri yapısı seçmeye yönelik bu üç adımlı yaklaşım, tasarım sürecinin veri merkezli olarak değerlendirilmesidir. Veriler ve bunlar üzerinde gerçekleştirilecek işlemler, bu verilerin temsil edilmesi ve bu temsilin uygulanmasını içerir.

Arama, veri kayıtlarının eklenmesi ve veri kayıtlarının silinmesi gibi belirli temel işlemler üzerindeki kaynak kısıtlamaları veri yapısı seçim sürecini yönlendirir. Bir veri yapısı seçmeniz gerektiğinde kendinize aşağıdaki üç soruyu sormalısınız:

Tüm veri öğeleri başlangıçta veri yapısına mı eklendi, yoksa eklemeler diğer işlemlerle serpiştirildi mi? Statik uygulamalar (verilerin başlangıçta yüklendiği ve hiçbir zaman değişmediği) verimli bir uygulama elde etmek için tipik olarak dinamik uygulamalardan daha basit veri yapıları gerektirir.

Veri öğeleri silinebilir mi? Eğer öyleyse, bu muhtemelen uygulamayı daha karmaşık hale getirecektir.

Tüm veri öğeleri iyi tanımlanmış bir sırada mı işleniyor yoksa belirli veri öğelerinin aranmasına izin veriliyor mu? "Rastgele erişim" araması genellikle daha karmaşık veri yapıları gerektirir.

Maliyetler ve Avantajlar

Her veri yapısının maliyetleri ve avantajları vardır. Her durum için aynı veri yapısının diğerinden daha iyi olduğu doğru değildir. Bir veri yapısı veya algoritması her bakımdan diğerinden üstünse, daha düşük olan genellikle çoktan unutulmuş olacaktır. Burada sunulan hemen hemen her veri yapısı ve algoritma ilgili problem için en iyi ihtimal olduğu örneklerde kullanılmıştır. Bazı örnekler sizi şaşırtabilir.

Bir veri yapısı, depoladığı her veri öğesi için belirli bir miktarda alan, tek bir temel işlemi gerçekleştirmek için belirli bir süre ve belirli bir miktarda programlama çabası gerektirir. Her problemin kullanılabilir alan ve zaman kısıtlamaları vardır. Bir problemin her çözümü, temel işlemleri göreceli bir oranda kullanır ve veri yapısı seçim süreci bunu hesaba katmalıdır. Ancak problemin özelliklerinin dikkatli bir analizinden sonra çözüm için en iyi veri yapısını belirleyebilirsiniz.

Örnek 1

Bir banka gerçekte müşterilerin yapacağı birçok işlem türünü destekler. Ancak burada; müşterilerin hesap açmak, hesap kapatmak, para eklemek veya hesaplardan para çekmek istediği basit bir modeli inceleyelim. Bu sorunu iki farklı düzeyde ele alabiliriz: (1) bankanın müşterileriyle etkileşimlerinde kullandığı fiziksel altyapı ve iş akışı süreci gereksinimleri ve (2) hesapları yöneten veritabanı sistemi gereksinimleri.

Genel olarak bir müşteri, sık olarak hesaba erişir fakat daha az sıklıkta hesap açar ve kapatır. Müşteriler, hesaplar oluşturulurken veya silinirken dakikalarca bekleyebilir, ancak para yatırma veya çekme gibi bireysel hesap işlemlerinin kısa bir sürede bitmesini bekler. Bu gözlemler, problemdeki zaman kısıtlamaları için resmi olmayan özellikler olarak kabul edilebilir.

Bankaların iki katmanlı hizmet sağlaması yaygın bir uygulamadır. Vezneciler veya otomatik vezne makineleri (ATM'ler), müşterilerin hesap bakiyelerine ve para yatırma ve çekme gibi güncellemelere erişimini destekler. Hesap açma ve kapama işlemlerini yürütmek için genellikle (kısıtlı saatlerde) özel hizmet temsilcileri sağlanır. Vezne ve ATM

işlemlerinin kısa sürmesi beklenir. Bir hesabı açmak veya kapatmak çok daha uzun sürebilir (müşterinin bakış açısından belki bir saate kadar).

Veri tabanı açısından bakıldığında, ATM işlemlerinin veri tabanını önemli ölçüde değiştirmediğini görüyoruz. Para eklenir veya çıkarılırsa, bu işlemin bir hesap kaydında depolanan değeri değiştirir. Veri tabanına yeni bir hesap eklemenin birkaç dakika sürmesine izin verilir. Bir hesabın silinmesi için herhangi bir zaman kısıtlaması yoktur, çünkü müşterinin bakış açısından önemli olan tek şey, tüm paranın iade edilmesidir (para çekme işlemine eşdeğerdir). Banka açısından hesap kaydı, mesai saatlerinden sonra veya aylık hesap döngüsünün sonunda veri tabanı sisteminden kaldırılabilir.

Müşteri hesaplarını yöneten veri tabanı sisteminde kullanılacak veri yapısı seçimi düşünüldüğünde, silme maliyetiyle çok az ilgilenilen, ancak arama için yüksek verimli ve ekleme için orta derecede verimli bir veri yapısının kaynağı karşılaması gerektiğini görüyoruz. Kayıtlara benzersiz hesap numarasıyla erişilebilir (tam-eşleşme sorgusu olarak ta adlandırılır). Bu gereksinimleri karşılayan bir veri yapısı, ilerde inceleyeceğimiz hash tablodur. Hash tabloları, son derece hızlı tam-eşleşme aramasına izin verir. Değişiklik alan gereksinimlerini etkilemediğinde bir kayıt hızlı bir şekilde değiştirilebilir. Hash tabloları ayrıca yeni kayıtların verimli bir şekilde eklenmesini de destekler. Silme işlemleri de verimli bir şekilde desteklenebilse de, çok fazla silme, kalan işlemler için performansta bir miktar düşüşe neden olur. Bununla birlikte, sistemi en yüksek verimliliğe geri döndürmek için hash tablosu periyodik olarak yeniden düzenlenebilir. Bu tür bir yeniden düzenleme, ATM işlemlerini etkilememek için çevrimdışı olarak gerçekleşebilir.

Örnek 2

Bir şirket, Türkiye'deki iller, ilçeler ve kasabalar hakkında bilgi içeren bir veri tabanı sistemi geliştiriyor. Veri tabanında binlerce şehir ve kasaba var ve veritabanı programı, kullanıcıların belirli bir yer hakkında adıyla bilgi bulmasına izin vermelidir (tam-eşleşme sorgusunun başka bir örneği). Kullanıcılar ayrıca, konum veya nüfus büyüklüğü gibi nitelikler için belirli bir değer veya değer aralığıyla eşleşen tüm yerleri bulabilmelidir. Bu, bir aralık sorgusu olarak bilinir.

Makul bir veri tabanı sistemi, kullanıcının bekleme süresini en azda tutmak için sorguları yeterince hızlı yanıtlamalıdır. Tam-eşleşme sorgusu için birkaç saniye yeterlidir. Veri tabanının, sorgu spesifikasyonuna uyan birçok şehri döndürebilecek aralık sorgularını desteklemesi amaçlanıyorsa, tüm işlemin daha uzun sürmesine (belki bir dakika kadar) izin

verilebilir. Bu gereksinimi karşılamak için, aralıktaki tüm şehirleri tek tek şehirler üzerinde bir dizi işlem yerine toplu olarak işleyerek aralık sorgularını verimli bir şekilde işleyen işlemleri desteklemek gerekecektir.

Önceki örnekte önerilen hash tablo, aralık sorguları verimli olarak gerçekleştiremediği için şehir veri tabanımızı uygulamak için uygun değildir. İlerde göreceğimiz B+ tree, büyük veri tabanlarını, veri kayıtlarının eklenmesini ve silinmesini ve aralık sorgularını destekler. Bununla birlikte, veritabanı bir kez oluşturulduysa ve daha sonra hiç değiştirilmiyorsa, ilerde açıklanacağı gibi basit bir doğrusal indis uygulaması daha uygun olacaktır.