Algoritma Tasarım Teknikleri

- Bilgisayar bilimcileri, çok farklı sorunları çözseler bile birçok algoritmanın benzer fikirleri paylaştığını keşfettiler.
- Bir algoritma tasarlarken uygulanabilecek nispeten az temel teknik vardır.
- Bu teknikler ilerleyen bölümlerde detaylı olarak incelenecektir.
- Bu bölümde en yaygın algoritma tasarım tekniklerinden bahsedilecektir.

Algoritma Tasarım Teknikleri

- Tasarım tekniklerini açıklamak için, kablosuz bir telefonla neredeyse herkesi rahatsız eden çok basit bir sorunu ele alacağız.
- Telsiz telefonunuzun çaldığını, ancak ahizeyi evinizin herhangi bir yerine koyduğunuzu varsayalım.
- Telefonu nasıl bulursunuz?
- İşleri karmaşıklaştırmak için, evinize bir kucak dolusu bakkaliye ile girdiniz ve hava karanlık, bu yüzden yalnızca görme ile bulmanız pek mümkün değil.

Algoritma Tasarım Teknikleri

Exhaustive Search (Ayrıntılı Arama)

- Kapsamlı bir arama veya kaba kuvvet algoritması, belirli bir çözüm bulmak için olası her alternatifi inceler.
- Örneğin, çalan telefonu bulmak için kaba kuvvet algoritmasını kullandıysanız, sanki duyamıyormuşsunuz gibi telefonun çalmasını görmezden gelirsiniz ve evinizin her santimetrekaresinin üzerinden geçip telefonun orda olup olmadığını kontrol edersiniz.
- Muhtemelen çok şanslı olmadıkça telefonun çalması durdurmadan önce cevap veremezsiniz, ancak sonunda nerede olursa olsun telefonu bulmanız garanti edilirdi.

Exhaustive Search (Ayrıntılı Arama)

- BRUTEFORCECHANGE bir kaba kuvvet algoritmasıdır, bunlar tasarlanması ve anlaşılması en kolay algoritmalardır ve bazen biyolojideki bazı pratik problemler için kabul edilebilir şekilde çalışırlar.
- Genel olarak, kaba kuvvet algoritmaları, en küçük örnekler dışında herhangi bir şey için pratik olamayacak kadar yavaştır.
- Kaba kuvvet algoritmalarından nasıl kaçınılacağını veya bunları daha hızlı sürümlere nasıl dönüştürüleceğini tartışacağız.

Exhaustive Search (Ayrıntılı Arama)

Branch-and-Bound Algorithms (Dal-Sınır)

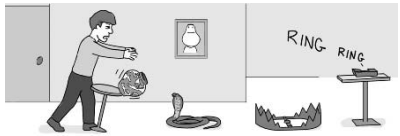
- Bazı durumlarda, bir kaba kuvvet algoritmasında çeşitli alternatifleri araştırırken, çok sayıda alternatifi atlayabileceğimizi keşfederiz, bu genellikle dallanma ve sınırlama veya budama olarak adlandırılan bir tekniktir.
- Birinci katı kapsamlı bir şekilde araştırdığınızı ve başınızın üzerinde telefonun çaldığını duyduğunuzu varsayalım.
- Bodrum katını veya birinci katı arama ihtiyacını hemen ortadan kaldırabilirsiniz.
- Üç saat süren şey, ekarte edebileceğiniz alan miktarına bağlı olarak artık yalnızca bir saat sürebilir.

Greedy Algorithms (Açgözlü)

- Çoğu algoritma, her yinelemede bir dizi alternatif arasından seçim yapan yinelemeli prosedürlerdir.
- Örneğin, bir kasiyer, kağıt dan metal paraya değişim problemini vermesi gereken bir dizi karar olarak görebilir: hangi metal paranın önce, hangisinin ikinci olarak kullanılması gerektiği vb.
- Bu alternatiflerden bazıları doğru çözümlere yol açabilirken diğerleri olmayabilir.
- Açgözlü algoritmalar, her yinelemede "en çekici" alternatifi, örneğin mümkün olan en büyük metal parayı seçer.
- Değişiklik yapmak için 1 tl, sonra 50 kuruş, sonra 25 kuruş vb. kullanır.
- Elbette, bu açgözlü stratejinin genelleştirilmesinin, bazı yeni metal para türleri eklendiğinde yanlış sonuçlar vereceği açıktır.

Greedy Algorithms (Açgözlü)

- Telefon örneğinde, karşılık gelen açgözlü algoritma, siz onu bulana kadar telefonun çaldığı yönde yürümek olacaktır.
- Buradaki sorun, sizinle telefon arasında onu bulmanızı engelleyen bir duvar (veya pahalı ve kırılgan bir vazo) olabileceğidir.
- Ne yazık ki, bu tür zorluklar çoğu gerçekçi problemde sıklıkla ortaya çıkar.
- Çoğu durumda, açgözlü bir yaklaşım çözüm olarak görünecek, ancak bir şekilde yanlış olacaktır.



Dynamic Programming

- Bazı algoritmalar bir problemi daha küçük alt problemlere böler ve daha büyük olanın çözümünü oluşturmak için alt problemlerin çözümlerini kullanır.
- Bu işlem sırasında, alt problemlerin sayısı çok büyük hale gelebilir ve bazı algoritmalar aynı alt problemi tekrar tekrar çözerek çalışma süresini gereksiz yere artırabilir.
- Dinamik programlama, zaten bildiğiniz değerlerin yeniden hesaplanmasını önlemek için hesaplamaları düzenler ve bu da büyük ölçüde zaman kazandırabilir.
- Telefonun Çalan problemi dinamik bir programlama çözümüne uygun değildir, bu yüzden tekniği açıklamak için farklı bir problemi ele alınacaktır.

Dynamic Programming

- Taş oyunu: İki oyuncu ve on taş içeren iki yığın düşünün.
- Her turda bir oyuncu tek bir yığından bir taş veya her iki yığından bir taş alabilir.
- Taşlar alındıktan sonra oyundan çıkarılır.
- Son taşı alan oyuncu oyunu kazanır.
- Telefona cevap vermek yerine, Taş oyununu iki yığın taşla oynamaya karar verdiğinizi varsayalım, her birinde on tane taş var.
- İlk hareketi siz yapacaksınız, kim kazanır?

Dynamic Programming

- 10+ 10 oyun için kazanan stratejiyi bulmak için aşağıda gösterilen R diyebileceğimiz bir tablo oluşturabiliriz.
- Her bir yığında 10 taş olan bir problemi çözmek yerine, n ve m'nin keyfi olduğu bir yığında n taş ve diğerinde m taş (n + m oyunu) olan daha genel bir problemi çözeceğiz.
- Oyuncu 1 her zaman 5 + 6 oyununu kazanabilirse, R5,6 = W olarak işaretlenir,
- Oyuncu 1'in her zaman doğru hamleleri yapan bir oyuncuya karşı kazanma olasılığı yoksa, R5,6 = L olarak işaretlenir.
- Rasgele bir n ve m için Rn, m'yi hesaplamak zor görünüyor, ancak daha küçük değerler üzerine inşa edebiliriz.
- Bazı oyunlar, özellikle R0,1, R1,0 ve R1,1, Oyuncu 1 için açıkça kazanılan oyunlardır.
- Çünkü ilk hamlede Oyuncu 1 kazanabilir.
- Böylece (1, 1), (0, 1) ve (1, 0) girişlerini W olarak doldurulur.

Dynamic Programming

	0	1	2	3	4	5	6	7	8	9	10
0		W									
1	W	W									
2											
3											
4											
5											
6											
7											
8											
9											
10											

Dynamic Programming

- (0, 1), (1, 0) ve (1, 1) girişleri doldurulduktan sonra diğer girişler doldurulmaya çalışılabilir.
- Örneğin, (2, 0) durumunda, Oyuncu 1'in yapabileceği tek hamle, zaten bildiğimiz gibi, rakibi için kazanan bir durum olan (1, 0) durumuna yol açar.
- Benzer bir analiz (0, 2) durumu için de geçerlidir ve aşağıdaki sonucu verir:

	0	1	2	3	4	5	6	7	8	9	10
0		W	L								
1	W	W									
2	L										
3											
4											
5											
6											
7											
8											
9											
10											

Dynamic Programming

	0	1	2	3	4	5	6	7	8	9	10
0		W	L	W	L	W	L	W	L	W	L
1	W	W	W	W	W	W	W	W	W	W	W
2	L	W	L	W	L	W	L	W	L	W	L
3	W	W	W	W	W	W	W	W	W	W	W
4	L	W	L	W	L	W	L	W	L	W	L
5	W	W	W	W	W	W	W	W	W	W	W
6	L	W	L	W	L	W	L	W	L	W	L
7	W	W	W	W	W	W	W	W	W	W	W
8	L	W	L	W	L	W	L	W	L	W	L
9	W	W	W	W	W	W	W	W	W	W	W
10	L	W	L	W	L	W	L	W	L	W	L

Divide-and-Conquer Algorithms (Böl ve Yönet)

- Büyük bir problemin çözülmesi zor olabilir, ancak yarısı boyutta olan iki problem önemli ölçüde daha kolay olabilir.
- Bu durumlarda, böl ve fethet algoritmaları tam da bunu yaparak başarılı olur: problemi daha küçük alt problemlere bölerek, alt problemleri bağımsız olarak çözer ve alt problemlerin çözümlerini orijinal problemin bir çözümünde birleştirir.
- Durum genellikle bundan daha karmaşıktır ve bir problemi alt problemlere böldükten sonra, bir böl ve yönet algoritması genellikle bu alt problemleri daha da küçük alt problemlere böler ve artık ihtiyaç duymadığı bir noktaya ulaşıncaya kadar böyle devam eder.
- Birçok böl ve yönet algoritmasındaki kritik bir adım, alt problemlere yönelik çözümlerin daha büyük bir problem için bir çözümde yeniden birleştirilmesidir.
- Çoğu zaman, bu birleştirme adımı önemli miktarda zaman tüketebilir.

Machine Learning

- Telefon arama sorununa bir başka yaklaşım, bir yıl boyunca telefonu nerede bıraktığınız hakkında istatistik toplamak ve telefonun çoğu zaman nerede olma eğiliminde olduğunu öğrenmektir.
- Telefon zamanın %80'inde banyoda, %15'inde yatak odasında ve %5'inde mutfakta bırakılsaydı, aramaya banyoda başlamak mantıklı bir zaman tasarrufu stratejisi olurdu.
- Yatak odasına devam edin ve mutfakta bitirin.
- Makine öğrenimi algoritmaları genellikle stratejilerini önceden toplanan verilerin hesaplamalı analizine dayandırır.

Randomized Algorithms

- Eğer bir bozuk paranız varsa, telefonu aramaya başlamadan önce, aramaya tura gelirse birinci katta mı yoksa yazı gelirse ikinci katta mı başlamak istediğinize karar vermek için kullanabilirsiniz.
- Bir zarınız varsa, ikinci kata karar verdikten sonra, ikinci kattaki altı odadan hangisinde aramanıza başlayacağınıza karar vermek için onu kullanabilirsiniz.
- Bozuk para ve zar atmak, telefonu aramak için eğlenceli bir yol olsa da, kesinlikle yapılması gereken sezgisel bir şey değildir ve size deterministik bir algoritmaya göre algoritmik bir avantaj sağlayıp sağlamadığı hiç de net değildir.
- Rastgele algoritmaların pratik problemleri çözmeye nasıl yardımcı olduğunu ve neden bazılarının deterministik algoritmalara göre rekabet avantajına sahip olduğunu daha sonra işlenecektir.

Çözülebilen ve Çözülemeyen Problemler

- Algoritmaların karmaşıklıklarına göre kategorize edilebileceğini gördük.
- Bilgisayar bilimcileri, sorunların aynı zamanda içsel karmaşıklıklarına göre sınıflandırılabileceğini keşfettiler.
- Bir n elemanlı kümenin her alt kümesini listelemek gibi bazı problemlerin üstel zaman gerektirdiği ortaya çıktı - ne kadar akıllı olursa olsun hiçbir algoritma sorunu üstel zamandan daha kısa sürede çözemez.
- Bir tam sayılar listesinin sıralanması gibi diğer sorunlar, yalnızca polinom zamanı gerektirir.
- Polinom problemleri ile üstel problemler arasında bir yerde NP-tam problemler adı verilen özellikle önemli bir problem kategorisi vardır.
- Bunlar, henüz bu sorunlardan herhangi biri için polinom zaman algoritması bulunamadığından, oldukça zor görünen problemlerdir.
- Bununla birlikte, hiç kimse bu problemler için polinom-zaman algoritmalarının imkansız olduğunu kanıtlayamaz, bu yüzden hiç kimse bu problemlerin gerçekten verimli bir şekilde çözülebilir olma olasılığını göz ardı edemez.
- NP-tam sorununun bilhassa ünlü bir örneği, biyolojide çok çeşitli pratik uygulamalara sahip olan Gezgin Satıcı problemidir.

Çözülebilen ve Çözülemeyen Problemler

- NP-tam problemlerin kritik özelliği, eğer bir NP-tam problemi bir polinom-zaman algoritması ile çözülebilirse, tüm NP-tam problemleri aynı algoritmanın küçük modifikasyonları ile çözülebilmesidir.
- Bunun henüz bulamadığı gerçeği, onun var olmayabileceğini gösteriyor.
- Ancak, bu henüz matematiksel olarak kanıtlanmamıştır.
- Binlerce algoritmik problemin aslında kılık değiştirmiş Gezgin Satıcı problemlerinin örnekleri olduğu ortaya çıktı.
- Bu problemler NP-tam problemler sınıfını oluşturur.