Algoritmalara Giriş 6 Ekim 2005
Massachusetts Institute of Technology 6.046J/18.410J
Profesörler Erik D. Demaine ve Charles E. Leiserson Kitapçık 11

Pratik Ara Sınav 1 Çözümleri

Problem -1. Yinelemeler

Aşağıdaki yinelemeleri, sıkı Θ-simgelemi sınırlarını vererek çözün. Cevabınızı açıklamanıza gerek yok, ancak vereceğiniz bilgiler gidiş yolundan puan almanızı sağlayabilir.

(a) $T(n) = T(n/3) + T(n/6) + \Theta(n^{\sqrt{\lg n}})$

Çözüm: Burada Ana Metod direkt olarak uygulanamaz. Ama,

$T(n) \le S(n) = 2T(n/3) + \Theta(n^{\sqrt{\lg n}})$

Şimdi bunu elde etmek için Ana Metodun 3. Şıkkını kullanın:

$T(n) \le S(n) = \Theta(n^{\sqrt{\lg n}})$

Ayrıca,

$T(n) = O(n^{\sqrt{\lg n}})$

olduğundan alt sınır oldukça nettir.

(b) $T(n) = T(n/2) + T(\sqrt{n}) + n$

Çözüm: Ana metod burada da direkt olarak uygulanamaz. Ama $\sqrt{n}$ n/2'den oldukça küçüktür, bu nedenle küçük olan terimi yok sayar ve yanıtın T(n) = Θ(n) olduğunu tahmin ederiz. Yerine koyma ile sağlamasını yaparız.

(c) $T(n) = 3T(n/5) + \lg^2 n$

Çözüm: Ana metodun, birinci durumundan faydalanarak,

$T(n) = \Theta(n^{\log_5(3)})$

(d) $T(n) = 2T(n/3) + n \lg n$

Çözüm: Ana Metodun 3. durumundan faydalanırsak,

$T(n) = \Theta(n \lg n)$

(e) $T(n) = T(n/5) + \lg^2 n$

Çözüm: Ana metodun 2. durumu ile
$T(n) = \Theta(\lg^3 n)$

(f) $T(n) = 8T(n/2) + n^3$

Çözüm: Ana metodun 2. durumundan faydalanarak
$T(n) = \Theta(n^3 \log n)$

(g) $T(n) = 7T(n/2) + n^3$

Çözüm: Ana metodun 3. durumundan faydalanırsak, T(n) = $\Theta(n^3)$

(h) $T(n) = T(n-2) + \lg n$

Çözüm:

$T(n) = \Theta(n \log n)$

Burada,

$\sum_{i=1}^{n/2} \lg 2i \geq \sum_{i=1}^{n/2} \lg i \geq (n/4)(\lg n/4) = \Omega(n \lg n)$

Üst sınır için, T(n)≤S(n)'dir. S(n)= S(n-1) + lg n olduğunda, O(n lg n) üst sınırımız olur.

Problem -2. Doğru veya Yanlış

Aşağıdaki ifadelerle ilgili, **D** veya **Y**'yi daire içine alarak, ifadelerin doğru mu yanlış mı olduğunu belirtin ve cevabınızı kısaca açıklayın. Anlatımınız ne kadar açık olursa, o kadar yüksek not alırsınız, ama kısa anlatmanız da önemli. Açıklama yapmadığınız takdirde, doğru cevaptan puan alamayacaksınız.

(a) D Y Asimptotik olarak pozitif olan bütün f(n)'ler için; f(n) + o(f(n))= Θ(f(n))'dir.

Çözüm: Doğru. f(n) + o(f(n)), Ω(f(n))'dir. g(n) Є o(f(n)) olsun. c'nin 0'dan büyük olduğu bütün değerler için ve bazı n_0'dan büyük bütün n'ler için, g(n)≤c(f(n))'dir.
Böylece g(n) = O(f(n))'e eşit olduğuna göre, f(n) + o(f(n)) = O(f(n))'dir.
Yani f(n) + o(f(n)) = Θ(f(n))'dir.

(b) D Y En kötü durumda ve beklenen durumdaki koşma süreleri, rastgele algoritmalar için, sabit faktörlere bağlıdır.

Çözüm: Yanlış. .. Rastgele çabuk sıralamanın en kötü durum koşma süresi $\Theta(n^2)$ve beklenen koşma süresi Ɵ(n lgn)'dir.

(c) D Y {A, B, C, D} evrenindeki anahtarları, aşağıdaki tabloya göre {0, 1, 2} aralığına eşlemleyen, H={ h_1, h_2, h_3} tanımlı 3 kıyım fonksiyonunun koleksiyonu evrenseldir.

x	$h_1(x)$	$h_2(x)$	$h_3(x)$
A	1	0	2
B	0	1	2
C	0	0	0
D	1	1	0

Çözüm: Doğru. U anahtarlar evrenini, m tane yuvaya eşlemleyen bir *H* kıyım ailesi, eğer x, y Є U olan farklı anahtar çiftleri için, h(x) = h(y) ise ve h Є *H* kıyım fonksiyonlarının sayısı |*H*|/m olursa *evrenseldir*.
Burada, |H| = 3 ve m = 3'tür. 4 farklı anahtarın her bir çifti için, bir çarpışmaya neden olacak kıyım fonksiyonu vardır. Yukarıdaki tablodan;

h(A) = h(B) , sadece h_3 için 2. yuvaya kıyılırlar.
h(A) = h(C) , sadece h_2 için 0. yuvaya kıyılırlar.
h(A) = h(D) , sadece h_1 için 1. yuvaya kıyılırlar.
h(B) = h(C) , sadece h_1 için 0. yuvaya kıyılırlar.
h(B) = h(D) , sadece h_2 için 1. yuvaya kıyılırlar.
h(C) = h(D) , sadece h_3 için 0. yuvaya kıyılırlar.

Problem -3. Kısa cevaplar

Aşağıdaki sorulara, kısa ama tam cevaplar verin.

(a) Herhangi bir karşılaştırmaya dayalı sıralama algoritması, koşma süresini sabit bir çarpandan daha fazla değiştirmeyecek şekilde kararlı olabilir.

Çözüm: Karşılaştırmaya dayalı bir sıralama algoritmasını kararlı yapabilmek için, bütün elemanları dizilimdeki asıl pozisyonlarına etiketleriz. Eğer A[i] = A[j] ise, elemanın pozisyonuna karar vermek için i ve j'yi kıyaslarız. Bu koşma süresini en fazla 2 kat arttırır.

(b) Kıyaslama modelinde, aşağıdaki özelliklerin ikisinin de olması durumunda Öncelikli Kuyruğun olamayacağını tartışın.

EXTRACT-MIN Θ(1) sürede koşacak.
BUILD-HEAP Θ(n) sürede koşacak.

Çözüm:

Eğer böyle bir Öncelikli Kuyruk var olsaydı, sıralamayı BUILD-HEAP(Θ(n))'i yürütür ve sonra n kere en küçüğü bularak (n. Θ(1) = Θ(n)) yapabilirdik. Bu durumda algoritmamız Θ(n) sürede çalışırdı ve bu da, kıyaslamaya dayalı sıralama algoritmalarının alt sınırı Θ(nlgn)'e uymazdı.

(c) A[1]'in en büyük anahtar olduğu A[1,2,...,n] dizilimi içindeki bir yığın (en büyük yığın) için, aşağıdaki yordamı uygulayacak ve aynı zamanda da en büyük yığın olma özelliğini koruyacak sözde kodu yazın.

$\textsc{Decrease-Key}(i, \delta)$ - A[i] anahtarının değerini δ kadar küçültür. δ'nın 0'dan büyük veya eşit olduğunu varsayın.

Çözüm:

$\textsc{Decrease-Key}(i, \delta)$
  $A[i] \leftarrow A[i] - \delta$
  $\textsc{Max-Heapify}(A, i)$

(d) n tane farklı tamsayıdan (bazıları eksi değerli olabilir) oluşan sıralı A dizilimi için;

$1 \le i \le n$ ve A[i]=i olan bir i dizinini (varsa) bulmak için bir algoritma yazın. Eğer birden fazla buna benzer dizin varsa, algoritmanın herhangi birini vermesi yeterlidir .

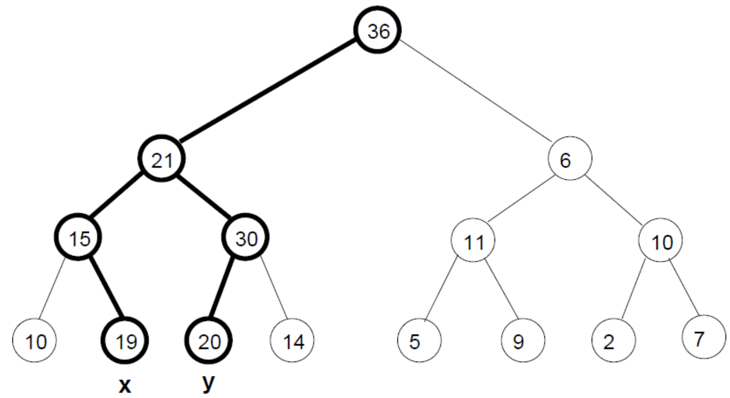
Çözüm:

Buradaki önemli gözlem, eğer A[j] > j ise ve A[i] = i ise, i < j demektir. Benzer şekilde eğer, A[j] < j ise ve A[i] = i ise, i > j demektir. Dolayısıyla, dizilimin ortasındaki elemana bakarsak, dizilimin yarısı elenebilir. Aşağıdaki algoritma, (INDEX-SEARCH) ikili aramaya benzer ve Θ(lgn) sürede koşar. Eğer herhangi bir cevap yoksa -1'i geri döndürür.

INDEX-SEARCH(A, b, e)
  if $(e > b)$
  return -1
  $m = \left\lceil \frac{e+b}{2} \right\rceil$
  if $A[m] = m$
    then return m
  if $A[m] > m$
    then return INDEX-SEARCH(A, b, m)
    else return INDEX-SEARCH(A, m, e)

Problem -4. h yüksekliğinde ve $n=2^h$ yapraklı bir ikili ağacın verildiğini düşünün. Bu ağacın her düğümü ve yaprağına bir v değeri (rastgele gerçek sayı) atanmış durumda. Eğer x bir yaprak ise, x'i ve x'in atalarının ve x'in büyük kümesini A(x) ile ifade ederiz. A(x) köke kadar gider. Benzer şekilde x ve y farklı yapraklarsa,

$$A(x, y) = A(x) \cup A(y)$$



A(x,y) koyu olarak gösterilmiştir.

f(x,y) = 19+15+21+36+20+30 = 141

f(x,y) fonksiyonunu A(x,y)'deki düğümlerin değerlerinin toplamı olarak tanımlayın. $f(x_0, y_0)$'ın en büyük olduğu x_0 ve y_0 yapraklarını bulmak için verimli bir algoritma (sözde kod) yazın. Algoritmanızın koşma süresi nedir?

Çözüm:

Bu sorunun farklı bir kaç çözüm yöntemi var. Sınıfta böl ve fethet'i çalıştığımızdan, burada böl ve fethet çözümünü veriyoruz. Ama bunun dışında da O(n), O(n lgn) ve O(n^2lgn) sürelerinde koşan, kaliteli başka algoritmalar da vardı. Bu algoritmaların değerleri sırasıyla 11, 9 ve 4 puandı. Doğru çözümleme 4 puan alacak.

Önce O(n lgn) süreli çözüme bakalım,sonra da bu çözümü nasıl O(n)'e çevireceğimizi gösterelim.

MAX1(z) özyinelemeli fonksiyonunu, $f(x)$'in en büyük değerini (bir düğümün atalarının toplamını) geri döndürmesi için tanımlıyoruz; bunu z'nin alt ağacındaki tüm x yaprakları için yapacak. Benzer şekilde, f(x,y)'nin en büyük değerini geri döndürecek MAX2(z)'yi, z alt-ağacındaki tüm x,y yaprak çiftleri için tanımlıyoruz. Kökte MAX2(z)'yi çağırmamız istenen sonucu verecektir.

Önce MAX1(z)'yi uygulayalım. En uzun yol ya z'nin sol alt ağacı, yada z'nin sağ alt ağacıdır; böylece aşağıdaki gibi basit bir böl ve fethet algoritması elde edilir.

MAX1(z)
1 **return** ($value(z) + \max\{\text{MAX1}(left[z]), \text{MAX1}(right[z])\}$)

MAX2(z) için, 3 olası çözüm olduğunu söyleyelim: 2 yaprak z'nin sağ alt ağacında, 2 yaprak z'nin sol alt ağacında, ya da 1 yaprak sağ, bir yaprak da sol alt ağaçtadır.

```
MAX2(z)
1  return (value(z) + max {MAX2(left[z]), MAX2(right[z]), MAX1(left[z]) + MAX1(right[z])})
```

Çözümleme: MAX1 için Ana Metodu uygulayarak aşağıdaki yinelemeyi elde ederiz.

$$\begin{aligned} T_1(n) &= 2T_1\left(\frac{n-1}{2}\right) + \Theta(1) \\ &= \Theta(n) \end{aligned}$$

MAX2 için ise Ana Metodun 2. durumu çerçevesinde;

$$\begin{aligned} T_2(n) &= 2T_2\left(\frac{n-1}{2}\right) + 2T_1\left(\frac{n-1}{2}\right) + \Theta(1) \\ &= 2T_2\left(\frac{n-1}{2}\right) + \Theta(n) \\ &= \Theta(n \lg n) \end{aligned}$$

elde ederiz.

O(n) çözümünü elde edebilmek için, bir MAXBOTH fonksiyonunu tanımlarız. Bu fonksiyon bir çifti geri döndürür (MAX1 ve MAX2'nin yanıtlarını). Bu basit değişiklikle yineleme MAX1'in aynısıdır.

Problem -5. Küçük çoklu kümelerin sıralanması

Bu problem için, n uzunluğunda, k farklı anahtardan oluşan A diziliminde,

$$k < \sqrt{n}$$

Amacımız bu dizilimi, Ω(n lgn)'den daha hızlı sıralayabilmek. Bunu 2 aşamada yapacağız. Birinci aşamada, A'daki k farklı anahtarı B *sıralı diziliminde* hesaplayacağız. İkinci aşamada, B dizilimini kullanarak, A dizilimini sıralayacağız.

k'nın bir sabit gibi çok küçük olabileceğini ve koşma sürenizin n kadar, k'ya da bağımlı olabileceğine dikkat edin. n öğenin anahtarlara ek olarak uydu verileri de var.

Örnek:

$$A = \left[5, 10^{10}, \pi, \tfrac{128}{279}, 10^{10}, \pi, 5, 10^{10}, \pi, \tfrac{128}{279}\right]$$ olsun.

Burada n= 10 ve k = 4 olur.
İlk aşamada,

$B = \left[\frac{128}{279}, \pi, 5, 10^{10}\right]$ hesaplarız.

İkinci aşamadan sonra sıralı dizilimimiz ise,

$$\left[\frac{128}{279}, \frac{128}{279}, \pi, \pi, \pi, 5, 5, 10^{10}, 10^{10}, 10^{10}\right]$$

olur. Hedefiniz, bu iki aşama için de verimli algoritmalar tasarlamak ve bunları çözümlemek. Unutmayın, algoritmanız ne kadar verimli olursa, alacağınız not da o kadar yüksek olur!

(a) Birinci aşama için, k farklı anahtarı olan k uzunluğundaki sıralı B dizilimini hesaplayacak bir algoritma tasarlayın. k'nın değeri algoritmaya girdi olarak verilmemektedir.

Çözüm:

Algoritma, farklı (tekrarlamayan) elemanları, B dizilimine eklerken, ara basamaklarda, dizilimi sıralı tutar. i= 1,2,...,n için, A[i] elemanı ikili arama ile B diziliminde aranır. Eğer A[i], B'de yer alıyorsa, araya yerleştirmeye gerek olmaz. Aksi halde ikili arama, bu elemanın dizilime eklenmesi gereken konumu da B'nin sıralı düzenini koruyarak bulur. B'nin içindeki, o konumun sağında olan tüm elemanlar, A[i]'ye yer açmak için bir konum kaydırılırlar.

(b) (a) bölümündeki algoritmanın çözümlemesini yapın.

Çözüm:

B diziliminde, A diziliminin her elemanının ikili araması, O(lg k) kadar süre gerektirir, çünkü B'nin boyutu en fazla k'dir. Bu da toplam da O(n lg k)'ya mal olur. Ayrıca her yeni elemanın B'de yerine yerleştirilmesi, tam olarak k sayıda işlem gerektirir, bu da O (1 + 2 + ... + k)'ya, yani O(k^2)'ye mal olur. Eğer, k< $\sqrt{n}$ ise, algoritmanın toplam maliyeti O [n lgk+ k^2] = O (n lgk) olur.

(c) İkinci aşama için bir algoritma tasarlayın; yani bölüm (a)'da yarattığınız B dizilimini kullanarak verilen bir A dizilimini sıralayın. Öğelerin uydu verileri olduğundan, belirli anahtarları olan elemanları sayıp ve kopyalamanın yeterli olmayacağına dikkat edin.

İpucu: Sayma sıralamasını uyarlayın..

Çözüm:

Sayma sıralamasında yaptığımız gibi bir C dizilimi oluşturun. C[i], A'da, değerleri, B[i]'den daha küçük olan elemanları içersin. Sayma sıralaması, olduğu gibi kullanıldığında, A[i] bir tam sayı olduğundan çalışmayacaktır.. Ayrıca, A[i] çok büyük bir tamsayı değeri de olabilir (giriş aralığımızın bir sınırlaması yok). Bu nedenle A[i], C dizilimimiz için geçersiz bir anahtar listesidir. Yapmak istediğimiz, A[i]'nin değeri için uygun bir tümlenik "etiket" atamaktır. Burada seçtiğimiz etiket, problemin son kısmında hesapladığımız, B'nin içindeki A[i]'nin değerinin dizinidir..

Bu dizini nasıl buluruz? B'yi baştan sona tarayıp, A[i] değerini arayabilir, sonra da B'de A[i]'yi içeren dizini geri döndürebiliriz. Bu O(k) kadar süre alır. Ancak, B zaten sıralı olduğundan, ikili aramayı kullanarak bu maliyeti O(lg k)'ya indirebiliriz. BINARY-SEARCH(S,x), S dizilimini ve x değerini girdi olarak alan, S[i]=x olduğundaki i değerini döndüren fonksiyon olsun. Sayma sıralamasının yeniden düzenlenmiş hali aşağıdadır; yeniden düzenlenen satırlar kalın fontla gösterilmiştir..

```
COUNTING-SORT(A)
/* Uses Arrays C[1..k], D[1..k], and A-out[1..n] */
For i = 1 to k do C[i] ← 0;                  /* Initialize */
For i = 1 to n do                            /* Count number of elements */
        Location ← BINARY-SEARCH(B, A[i]);
        C[Location] ← C[Location] +1;
D[1] ← C[1];
For j = 2 to k do                            /* Build cumulative counts */
        D[j] ← D[j − 1] + C[j];
For i = n downto 1 do                        /* Construct Sorted List A-Out */
        Location ← BINARY-SEARCH(B, A[i]);
        Out-Location ← D[Location];
        D[Location] ← D[Location] −1;
        A-out[Out-Location] ← A[i];
Output(A-out);
```

(d) (c) bölümündeki algoritmayı çözümleyin.

Çözüm:

Düzenlenmiş sayma sıralamasının çözümlemesi, şu şekilde parçalara ayrılabilir.

Birinci Döngü : O(k)

İkinci Döngü : O(n) döngü, her döngü, k boyutunda bir dizilimde ikili arama gerektirdiğinden, toplam iş : O(n lg k)

Üçüncü Döngü : O(k)

Dördüncü Döngü : O(n) döngü, her döngü, k boyutunda bir dizilimde ikili arama gerektirdiğinden, toplam iş : O(n lg k)

Koşma süresi, 2. ve 4. döngüler tarafından domine edildiğinden, toplam koşma süresi O(n lg k)'dır.