
Problem Seti 7

Okumalar: Bölüm 15, 16.1 – 16.3, 22.1 ve 23

Problem 7-1'i çözmeniz zorunludur. Bunun çözümünü vermemeniz yarı yıl notunuza önemli ölçüde olumsuz biçimde yansır. Hem egzersizler hem de problemler çözülecek, ama sadece problemler teslim edilecektir. Egzersizler ders materyalini hazmettirmek amacıyla hazırlanmıştır. Her ne kadar egzersiz çözümlerini teslim etmeyeceksiniz de egzersizdeki konulardan sorumlu olacaksınız.

Her sayfanın üstüne adınızı, dersin kod numarasını, problemin numarasını, etüt bölümünüzü ve ortak çalışma yaptığınız arkadaşlarınızın isimleriyle tarihleri yazın. Lütfen çözümlerinizi zımbalayın ve üç delikli kağıtta teslim edin.

Sıksık bir problem için "bir algoritma bulun" isteğiyle karşılaşacaksınız. Bu konudaki yanıtınız kısa bir makale şeklinde olmalıdır. Makalenin konu paragrafı, çözdüğünüz problem ve sonuçlarınızı özetleyecek şekilde düzenlenmelidir. Makalenizin ana yapısında aşağıdaki bilgiler verilmelidir:

1. Algoritmanın İngilizce açıklaması ve eğer faydalı olacaksa sözde kodu..

2.Algoritmanızın nasıl çalıştığını gösteren en az bir işlenmiş örnek veya şekil.

3.Algoritmanın doğruluğunun kanıtı (veya göstergesi).
4. Algoritmanın koşma zamanının çözümlemesi.

Amacınız iletişim kurmaktır.Tam not sadece iyi açıklanan doğru yanıtlara verilecektir, net olmayan açıklamalar daha düşük notlandırılacaktır.

Egzersiz 7-1. Kitaptaki 15.4-3 nolu egzersiz (356. Sayfa)
Egzersiz 7-2. Kitaptaki 16.1-3 nolu egzersiz (379. Sayfa)
Egzersiz 7-3. Kitaptaki 16.3-2 nolu egzersiz (384. Sayfa)
Egzersiz 7-4. Kitaptaki 22.1-5 nolu egzersiz (530. Sayfa)
Egzersiz 7-5. Kitaptaki 23.1-5 nolu egzersiz (566. Sayfa)
Egzersiz 7-6. Kitaptaki 23.2-4 nolu egzersiz (574. Sayfa)
Egzersiz 7-7. Kitaptaki 23.2-5 nolu egzersiz (574. Sayfa)

Problem 7-1. Edit distance (Biçimlendirme mesafesi)

Bu problemde biçimlendirme mesafesini hesaplamak için bir program yazacaksınız. Bu çözülmesi zorunlu bir problem. Bunu yapıp teslim etmezseniz yarı-yıl notunuzu önemli ölçüde olumsuz olarak etkileyecektir. Bu programlama ödevini bir an önce başlamanızı öneririz. Çünkü tüm detayları programın içine doğru yerleştirmek, umduğunuzdan fazla zaman alabilir.

Birçok kelime işlemcisi ve anahtar sözcük arama motorunun bir yazım düzeltme özelliği vardır. Eğer bir *x* sözcüğünü yanlış yazarsanız, kelime işlemcisi veya arama motoru bir *y* düzeltmesi önerebilir. *y* düzeltmesi, *x*' e yakın bir sözcük olmalıdır. Yazımdaki 2 harf dizgisi arasındaki benzerliği ölçmenin bir yolu, ''edit distance'' yani biçimlendirme mesafesidir. Biçimlendirme mesafesi kavramı, başka alanlarda da yararlıdır. Örneğin; biyologlar,DNA veya protein dizileri arasındaki benzerliği, biçimlendirme mesafesi kullanarak belirtirler.

x [1.. m] ve *y [1.. n]* gibi 2 harf dizgisinin biçimlendirme mesafesi *x[1.. m]* dizgisinin *y[1..* n][1] dizgisine dönüştüren''dönüştürme işlemleri'' dizisinin en az maliyetli olanı olarak tanımlanır (aşağıda tanımlanmıştır).Dönüşüm işlemlerinin etkisini tanmlamak için ara sonuçları saklayan bir *z[1.. s]*dizgisi kullanırız. Dönüşüm dizisinin başında *s = m ve z[1.. s] = x[1.. m]* (yani biz *x[1.. m]* dizgisiyle başlarız). Dönüşüm dizisinin sonunda elimizde *s = n ve z[1.. s] = y[1.. n]* olmalıdır. (yani hedefimiz, *y[.. n]* dizgisine dönüşmektir*).* Dönüşüm boyunca z dizgisinin uzunluğu olan *s*' yi ve imleç konumu olan *i*' yi (yani *z* dizgisinin bir anahtar listesini) koruruz. Dönüşüm boyunca $1 \leq i \leq s+1$ değişmezi her zaman geçerlidir. (imleçin, *z* dizgisinin sonundan bir adım öteye gidebildiğine ve böylece dizginin sonuna ekleme yapabildiğine dikkat edin).

Her dönüşüm işlemi, *z* dizgisini, *s* boyutunu ve *I* imleç konumunu değiştirebilir. Her dönüşüm işleminin ilgili bir maliyeti vardır. Dönüşüm işlemleri dizisinin maliyeti, dizideki bağımsız işlemlerin maliyetlerinin toplamına eşittir. Biçimlendirme probleminin hedefi *x[1.. m]* 'yi *y[1.. n]*'ye dönüştürecek dönüşüm işlemleri dizisini, en az maliyetli olanını bulmaktır.

[1] Burada bir metin dizgisini, harflerin bir dizilimi olarak görüyoruz. Bağımsız harfler sabit zamanda işlenebilir

5 dönüşüm işlemi vardır:

İşlem	*Maliyet*	*Etki*
Left(sol)	0	*i*=1 ise birşey yapma, değilse i ← *i*-1
Right(sağ)	0	*i*=s+1 ise birşey yapma, değilse i ← *i*+1
Replace(değiştir)	4	*i* = s+1 ise birşey yapma, değilse imleçin altındaki harfi c karakteriyle değiştir ve z[*i*] ← c yaptıktan sonra *i'* yi arttır.
Delete(sil)	2	*i* = s+1 ise birşey yapma, değilse imleçin altındaki *c* harfini z [*i*..s] ← z[*i*+1..s+1] yaptıktan sonra *s*' yi azalt. İmleç konumu *i* değişmeyecek.
Insert(araya yerleştir)	3	*c* harfini, *s'* yi arttırarak, *z* dizgisinde araya yerleştir ve z[i+1..s] ← z[i..s-1] ile z[i] ← c yap; sonra da *i*' yi arttır.

Örnek olarak kaynak dizgisi "***algorithm***" i, hedef dizgisi "***analysis***" e dönüştürme yollarından biri Tablo 1' deki işlemler dizisidir; burada altı çizili harf ***i*** imleçinin konumunu gösterir. Tablo 1' deki çözüm tek çözüm değildir, ***algorithm***'i ***analysis***'e dönüştüren birçok dönüşüm işlemi dizisi vardır ve bunların bazıları daha fazla bazıları da daha az maliyet çıkarır.

İşlem T	*z*	Maliyet	*Toplam*
ilk dizgi	algorithm	0	0
sağ	algorithm	0	0
sağ	algorithm	0	0
y ile değiştir	alyorithm	4	4
s ile değiştir	alysrithm	4	8
i ile değiştir	alysiithm	4	12
s ile değiştir	alysisthm	4	16
sil	alysishm	2	18
sil	alysism	2	20
sil	alysis	2	22
sol	alysis	0	22
sol	alysis	0	22
sol	alysis	0	22
sol	alysis	0	22
sol	alysis	0	22
n'yi Ara.Yer.	anlysis	3	25
a'yı Ara.Yer.	analysis	3	28

Tablo 1: Algorithm' i analysis'e dönüştürmek.

(a) Algorithm'i analysis' e 'sol' işlemini kullanmadan da dönüştürmek mümkündür. Tablo 1 ile maliyeti aynı olan ancak 'sol' işlemini kullanmayan bir işlemler dizisi verin.

(b) Biçimlendirme mesafesi *d(x,y)* olan herhangi iki *x* ve *y* dizgisi için, *x*'i *y*' ye, *d(x,y)* maliyetiyle dönüştüren ve hiç 'sol' işlemi olmayan bir *S* dönüştürme işlemleri dizisi olduğunu tartışın.

(c) Biçimlendirme mesafesi *d(x,y)*' yi hesaplama probleminin en iyi alt yapıyı kullandığını gösterin. *(İpucu: x* ve *y*' nin tüm son takılarını ele alın. *)*

(d) *d(x, y)* 'nin biçimlendirme mesafesinin değerini *x* ve *y*' nin sontakıları cinsinden özyinelemeli olarak tanımlayın. Biçimlendirme mesafesinin nasıl örtüşme altproblemleri oluşturduğunu gösterin.

(e) *x[1.. m]* 'den *y[1.. n]'*ye biçimlendirme mesafesini hesaplayan bir dinamik programlama algoritmasını açıklayın. (Memolandırılmış özyinelemeli bir algoritma kullanmayın. Algoritmanız klasik, aşağıdan yukarıya, tablo yapısını destekleyen bir algoritma olsun.) Algoritmanızın koşma süresini ve alan gereksinimini çözümleyin.

(f) Algoritmanızı dilediğiniz dilde bir bilgiişlem programi olarak uygulayın.[2] Programınız dinamik programlama kullanarak x ve y gibi iki dizgi arasındaki *d(x, y)* biçimlendirme mesafesini hesaplamalı ve ilgili dönüşüm işlemleri dizisine Tablo 1 'deki stilde baskıya göndermelidir.

Programınızı aşağıdaki dizgiler üzerinde çalıştırın:

x = "electrical engineering", (elektrik mühendisliği)

y = "computer science". (bilgiişlem bilimi)

Programınızın kaynak kodunu sınıfın web sitesine gönderin ve kaynak kodunuzla sonuçlarınızın baskı kopyasını verin.

---

[2] Çözümler Java ve Python ile sağlanacaktır

Programınızdaki hataları ayıklamanıza yardımcı olmak için sınıf web sitesinde örnek Girdi ve Çıktı Tekstleri verilmiştir. Bu çözümler tek değildir: Aynı maliyeti olan başka dönüşüm dizileri de olabilir. Her zamanki gibi,bu problemin çözümünde işbirliği yapabilirsiniz ama, programı kendiniz yazmalısınız.

(g) Sınıf web sitesinde verilen 3 girdi dosyasıyla programınızı çalıştırın. Her girdi dosyası aşağıdaki 4 satırı içermektedir:

1. Dizgi *x*' deki *m* karakterlerinin sayısı.
2. Dizgi *x*.
3. Dizgi *y'* deki *n* karakterlerinin sayısı.
4. Dizgi *y.*

Her girdi için biçimlendirme mesafesi *d(x,y)*' yi hesaplayın. Problemin bu bölümü için dönüşüm işlemlerinin basılmış halini teslim etmeyin. (web'de aramadan tekstlerin kaynağını belirleyebilirsiniz).

(h) Eğer z, bir dizilim kullanılarak uygulanırsa, o zaman ''araya yerleştirme'' ve ''silme'' işlemleri için ϴ(n) süresi gereklidir. 5 dönüşüm işleminin her birine, *O(1)* sürede gerçekleştirecek uygun bir veri yapısı tasarlayın.

Problem 7-2. GreedSox

GreedSox, popüler bir baseball takımıdır ve sadece bir şeyle ilgilenir: para kazanma. Onlar sizi toplu bilet satışlarını arttırma konusunda danışman olarak kiralamışlar. Şu problemi farketmişler. Bir grup, bir maçı seyretmek istediğinde, grubun tüm üyeleri açık tribünde koltuk istiyor, yoksa gidiyorlar. Gruplar parçalanarak oturtulmadığından dolayı, açık tribün genellikle dolu olmuyor. Oturacak yerler oluyor fakat bütün grup için yeterli olmuyor. Bu durumda da grup oturtulamadığından, GreedSox zarar ediyor.

GreedSox; yeni bir oturma planı için sizin tavsiyenizi istiyor. İlk gelen ilk oturur politikası yerine, GreedSox önce büyük gruplara, sonra küçük gruplara ve en son da kişilere (yani 1' li gruplara) yer ayırmaya karar veriyor.

Size $G[1..m] = [g1, g_2, \ldots, g_m]$, gibi grup kümeleri veriliyor ve burada g_i grubun büyüklüğünü gösteren sayı. Açık tribünde *n* sayıda insanın oturabileceğini varsayın. ADMIT(i) 'nin grup i ' yi kabul ettiği ve REJECT(i)' nin de grup(i)' yi reddettiği aşağıdaki hırslı oturma algoritmasını düşünün.

```
SEAT(G[1..m],n)
1  admitted <— 0          (yerleştirilen)
2  remaining <— n         (kalan)
3  G <- SORT(G)           ▷ Grupları büyükten küçüğe sıralayın.
4  for i <— 1 to m
5      do if G[i] < remaining
6          then ADMIT (i)    (öyleyse yerleştir)
7               remaining <- - remaining — G[i]
8               admitted <—  admitted + G[i]
9          else REJECT (i)   (başkaysa reddet)
10 return admitted       (yerleştirildi döndür)
```

SEAT algoritması (koltuk algoritması) grupları önce boyutuna göre sıralıyor. Sonra gruplar arasında büyükten küçüğe bir döngüye girerek açık tribüncüleri alacak herhangi bir gruba yer veriyor. Kabul edilen insan sayısını çıkışa veriyor.

(a) GreedSox' ın sahipleri haklı, hırslı yerleştirme algoritması hızlı çalışıyor. Hırslı yerleştirme algoritmasının en az *k/2* kişiyi oturtmaya izin vermesi durumunda *G* ve *n* verildiğinde *k* kişinin tribune alınabileceğini gösterin.

(b) Maalesef SEAT algoritması mükemmel çalışmaz. SEAT' ın en iyi çözüm olmadığını ters bir örnekle gösterin; asimptotik olarak *n* büyüdükçe, hırslı yerleştirme ile en iyi yerleştirme arasındaki oran *1/2'* ye yaklaşır.

Sonuçlarınızı GreedSox' un sahiplerine sunduğunuzda aşağıdaki probleme dikkatinizi çekerler:bir bilgisayarın belleğindeki sayıların aksine, gerçek insanları yerlerinden oynatmak zordur. Özellikle kuyrukta bekleyen insanlar ''sıralanmaktan'' hoşlanmazlar. GreedSox' un sahipleri, sizden *G* kümesini değiştirmeyecek bir hırslı yerleştirme algoritması versiyonunu geliştirmenizi isterler. (*G*' nin salt-okunur bellekte depolandığını düşünebilirsiniz). Aşağıdaki algoritmayı öneriyorsunuz.

```
RESEAT (G[1 . . m], n)   (yeniden yerleştir)
 1  admitted  ← 0
 2  remaining  ← n
 3  for j  ←1 to |lg n]
 4      do for i  ← 1 to m
 5          do if G[i] ≥ n/2^j and G[i] ≤  remaining
 6              then ADMIT (i)
 7                   remaining  ← remaining - G[i]
 8                   admitted  ← admitted + G[i]
 9              else if G[i] > remaining
10                     then REJECT (i)  (öyleyse reddet)
11  return admitted
```

RESEAT algoritması (yeniden yerleştirme algoritması), grup listelerinden birkaç kez döngüye girer. Birinci döngüde boyutu en az *n/2* olan bir grubu kabul eder. İkinci döngüde boyutu en az *n/4* olan bir grubu kabul eder. Bu şekilde devam ederek, açık tribün dolana kadar, giderek daha küçük gruplara oturma yeri bulur. RESEAT işini bitirdiğinde yerleştirilen kişilerin sayısını çıkışa verir.

(c) G ve n verildiğinde en az *k* sayıda kişinin kabul edildiğini varsayın. RESEAT algoritmasının hala *k/2* kişiyi yerleştireceğini gösterin.

(d) RESEAT algoritmasının koşma süresi *O(m lg n)* ' dir.Eğer *k* kişi yerleştirilebiliyorsa, O(m) koşma süresinde en az *k/2* kişiyi yerleştirebilen yeni bir algoritma kuramlayın.