Gebze Technical University
Computer Engineering

CSE 321
Introduction To Algorithm Design

HOMEWORK 3

KADİR BULUT
121044005

Course Assistant: AHMET SOYYİĞİT

QUESTION 1

Devrim Arabaları

Subject of Devrim Arabaları is quite beautiful and extraordinary. Cemal Gürsel who is head of government was impressed in the negative direction in a automotive congress since foreign states belived that the Turks can not produce an automotive on their own. He gives orders for the production of the first native Turkish automotive and in this film contains stages of producing an automotive by a group of railway engineers . At first nobody believes that an automotive will be produced even most of the assigned engineers . Because they have a very short time like 130 days and they have to train until October 29 Republic Day . A large team is being formed and the team is starting to work quickly . They work day and night and face a lot of difficulties but they do not give up . There are many people who can not accept this development even bureaucrats . Because this development will make a big contribution to the development of Turkey and will upset supporters of foreign states . No one, including the media, accepts it despite the fact that the engineers are very enthusiastic about their work . Despite the big progresses , those who do not believe still seem to have done nothing . When the days go by quickly , they want to build one more automotive to make things difficult . Despite all these negativities, the team does not give up and accepts it , too . This is actually the story of the film . Why we can not be in a better place due to shallow thoughts … At the end, two automotives called the revolution are being produced . On Republic Day , they empty the cars' gasoline to turn success into success in front of the public . The president gets in the car and stops after the car is gone. The president is commenting : “ We produce cars with western head, but forget to put gas in the eastern head .” . This is indicated as failure in the media and malicious intentions are suppressing success . I wish the revolution could be produced in series I am sure that the position of our country could be very different . In the movie there was a sentence from the character of Latif : “ In Turkey no success will be unpunished . ”. I think this sentence can summarize of the whole film . This film is the best proof of how important success in Turkey . Success is not always appreciated , desired success is appreciated . I hope the days when success is fully appreciated are closer .

QUESTION 2

* Bu soruda brute force algoritmasını kullanmamız istendiğinden dolayı tüm ihtimalleri değerlendirmem yani oluşabilecek tüm durumları kontrol etmem gerekiyordu. Bu soru için toplamda 4 farklı fonksiyon geliştirdim. Bunlar ;

1-) perm : permütasyon fonksiyonu . Bu fonksiyonu kullanmamdaki amaç asistanların kaç farklı şekilde derslere yerleştirilebileceğini bulabilmek . Yani asistanlar dersleri kaç farklı sıralamada alabilir , bunu bulabilmekti . Bu fonksiyonlada asistanların farklı diziliş durumlarını bir listeyle alabiliyorum .

2-) comb: kombinasyon fonksiyonu. Bu fonksiyonu kullanmamdaki amaç aslında asistan sayısının ders sayısından fazla olma durumunda ki tüm ihtimalleri değerlendirebilmek . Çünkü asistan sayısının daha fazla olduğu durumlarda optimal çözüm elde edebilmek için ; ders sayısı kadar saatleri toplamı optimal olan asistanı derslere dağıtmak fazla kalan asistan(lar)ıda 6 saatlik diğer görevlere dağıtmam gerekiyordu . Bu fonksiyonlada asistan sayısının fazla olduğu durumda toplam asistanlardan ders sayısı kadar asistanı kaç farklı şekilde seçebileceğimi kontrol edip listede tutuyorum.

3-) find : yazmış olduğum helper bir fonksiyon. Bu fonksiyon ise gönderilen bir inputTable objesi için daha doğrusu iç içe geçmiş listeler arasında optimal çözüme sahip asistan sıralanışını ve optimal zamanı return ediyor. Permütasyon fonksiyonunu kullanarak asistanların tüm olası yerleşme durumlarını değerlendirip , buna göre aralarından optimal olanını döndürüyor.

4-) findOptimalAssistantship : soruda bizden istenen main fonksiyon . Asistan sayısının ders sayısından fazla olduğu durumlarda yukarıdaki fonksiyon yetersiz kalıyor . Çünkü asistanlardan ders sayısı kadar asistanın kaç farklı şekilde seçilebileceğinide kontrol etmek gerekiyor. Bu fonksiyonda asistan sayısından ders sayısı kadar kaç farklı şekilde asistan seçilebiliyorsa kombinasyon fonksiyonu yardımı ile seçip , yukarıdaki fonksiyona göndererek optimal bir list ve zaman elde ediyorum. Bunu tüm durumlar için uygulayınca da seçilen her asistan topluluğunun çözümlerinden gelen optimal sıralanışları ve zamanları listelerde tutup bunlar arasından tekrar en opitmal olanlarını seçiyorum . En sonunda da toplam zamanı ve sıralanışı düzenleyip return ediyorum.

Best Case Analysis

Yazmış olduğum algoritma için best case durum asistan sayısının ders sayısına eşit olduğu durumdur (RA = courses). Ders sayısı asistan sayısına eşit olduğundan kombinasyon fonksiyonunu hiç kullanmaz . Sadece permütasyon fonksiyonu yardımı ile sıralamaları bulur mevcut asistanlar için.

Permütasyon fonksiyonumun karmaşıklığı:
n elemanlı liste için n! kez çalışacağından $\Theta(n!)$ dir.

r sayıda asistan ve n sayıda dersten oluşan bir girdi için :
Bu durumda fonksiyonlarım içinde karmaşıklığı belirleyici kısımlar :

findOptimalAssistantship fonksiyonum sadece **find** helper fonksiyonumu çağırır.

```
for x in range(0,courses): →n kez çalışır . Dolayısı ile Θ(n)
    temp.append(x)
```

```
# get all conditions for RA's with permutation
allConditions=perm(temp) →perm. fonks.  karmaşıklığına eşittir. Dolayısı ile Θ(n!)
allSums=[] #to keep times

#calculate total times for every condition in the permutation of table
for i in allConditions: →perm. fonks.  karmaşıklığına eşittir. Dolayısı ile Θ(n!)
    tempSum=0
    for j in range(0,courses): →n kez çalışır . Dolayısı ile Θ(n)
        tempSum+=inputTable[j][i[j]]
    allSums.append(tempSum) #add sums to list
```

Best case durumunda karmaşıklığı belirleyici kısım iç içe iki for döngüsünün olduğu mavi renkli kısımdır . Doalyısı ile time complexity n.n! $\rightarrow \Theta(n.n!)$

Worst Case Analysis

Yazmış olduğum algoritma için worst case durum asistan sayısının ders sayısından fazla olduğu durumdur (RA > courses). Ders sayısı asistan sayısından fazla olduğundan kombinasyon fonksiyonu kullanılarak ders sayısı kadar asistan kaç farklı şekilde seçilebilir durumu kontrol edilir. Daha sonra ise tüm durumlar için asistanlar kaç farklı sekilde derslere atanabilir durumu kontrol edilerek optimal bir time ve atama durumu elde edilir.

Kombinasyon fonksiyonumun karmaşıklığı:
r sayıda asistan ve n sayıda dersten oluşan bir girdide r nin n li kombinasyonları için time complexity:

$\Theta\left(\frac{r!}{(r-n)!\cdot n!}\right)$ (burada n bir katsayı)= $\Theta(r^n)$

Bu durumda fonksiyonlarım içinde karmaşıklığı belirleyici kısımlar :

```
allCombins=comb(indexes,courses,l) →Θ(r^n)

#evaluate all conditions
for i in allCombins: →Θ(r^n)
    temp=[]
    for j in range(0,courses): →Θ(n) (karmaşıklığı find fonksiyonundan küçüktür)
        temp.append(inputTable[i[j]])
    temp=find(temp) # get a optimal solution for a situation Θ(r.r!) (*yukarıda
                                                            bulmuştuk karmaşıklığını)
    resultLists.append(temp[0]) # get optimal list and add result list as a sublist
    resultTimes.append(temp[1])  # get optimal time and add result list as a value
```

Worst case durumunda karmaşıklığı belirleyici kısımlar mavi renkli kısımlardır.
Worst case durumunda karmaşıklık → $\Theta(r^n)$. $\Theta(r.r!)$ → $\Theta(r^n.r.r!)$

QUESTION 3

* Bu soruda öncelikle soruda istenildiği gibi bir bfs algoritması geliştirdim. Yazdığım bu fonksiyon bir graph (soruda verilen map) ve bir key(vertex) alıyor parametre olarak . Ve bu graph üzerindeki vertexleri dolaşarak verilen key yani vertexden hangi vertexlere yol kurulabileceğini sıralanmış bir liste şeklinde return ediyor.

*Yazmış olduğum diğer ana fonksiyonda :

- Öncelikle bir yol tamiri için gerekli olan costun bir lab inşa etmek için gerekli costtan daha fazla olma durumunu kontrol ediyorum . Eğer öyleyse tüm vertexlerde birer lab inşa etmek daha minimum maaliyet elde etmemize sebep oluyor.

-Diğer durumda ise ; öncelikle map teki tüm keyleri yani tüm vertexleri bir list'e atayıp burada tutuyorum . Birde control değişkeni tutup son vertex yani max. vertexe gelip gelmediğimi kontrol ediyorum. Her vertexi tek tek dolaşarak ve yazdığım bfs algoritmasını kullanarak vertexden hangi vertexlere yol kurulabileceğini ayrı bir listede tutuyorum . Burada minimum cost için kontrol ettiğim iki şey var;

1-) n department arasında minimum (n-1) yolla bağlantı sağlanabilir.
2-) graph içerisinde birbiriyle bağlantısı olmayan sub-graph sayısı kadar lab kurulması gerekir.

Bu iki kurala göre vertexleri tek tek gezip bir nevi bir vertexden başlayıp kurabileceğim max vertexe sahip graphı elde edip , eleman sayısının bir eksiği kadar yol , 1 adet lab kurarak min maliyeti elde etmiş oluyorum.

Worst Case Analysis

Yazmış olduğum algoritmaya göre worst case durumu graphtaki tüm vertexlerin başlı başına birer lab inşa ettirmeyi gerektirmesi , yani vertexler arasında herhangi bir edge bulunmaması durumudur. Çünkü bu durumda yazdığım fonksiyonun tüm vertexleri tek tek kontrol ederek ilerlemesi gerekiyor. Diğer (edgelerin bulunduğu) durumlarda arama yaparken bir vertex için edgeleri bulduktan sonra bulduğu edgeleri ayrı birer vertex olarak değerlendirmesi gerekmiyor.

n vertexe (key) sahip bir girdi için ana fonksiyonumdaki belirleyici kısımlar:

```
control=0
while control<size: → döngü n kez dönecektir(diğer durumlarda daha az döner controlde dolayı )
    tempKey=vertexesList.pop(0) # get first vertex
    tempList=list(myBfs(mapOfGTU,tempKey)) # get edges with vertex → bfs fonksiyonum
    if control < max(tempList): # check that last vertex
        numLabs+=1
        numRoads+=(len(tempList)-1)
    control=max(tempList) # get second vertex(! other component graph's first vertex)
```

bfs fonksiyonumda belirleyici kısım (n vertex için) :

```
for i in range(1,size+1): → döngü n kez dönecektir
    tempList=graph[i]
    for j in tempList: → herhangi bir bağlantı olmadığından hiçbir vertex için bu döngü çalışmaz
        if visitVertexes.count(j)>0:
```

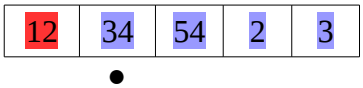
```
        if i not in visitVertexes:
            visitVertexes.append(i)
```

bfs fonksiyonum için çalışma zamanı →Θ (n) dir.
findMinimumCostToLabifyGTU fonksiyonum için çalışma zamanı :

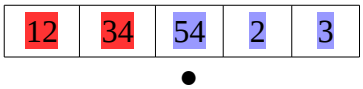
n . n →Θ (n^2) dir.

QUESTION 4

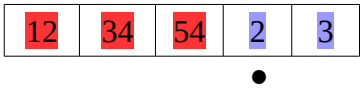
INSERTION SORT



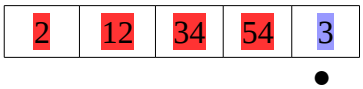
34 > 12 olduğundan yeri değişmeden kalır.



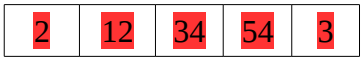
54>34 ve 54>12 olduğundan yeri değişmeden kalır.



2<12<34<54 olduğundan diğer elemanlar 1 er sağa kayarken , 2 ilk indexe yerleşir. (2<12<34<54)



3>2 ve 3<12<34<54 olduğundan 2 den sonraki elemanlar 1 er sağa kayarken 3 ikinci indexe yerleşir.



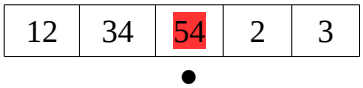
Ve sonuç olarak sıralanmış dizi elde edilir.

2	3	12	34	54

SHELL SORT

12	34	54	2	3

Diziyi sıralamak için gerekli atlama miktarı gap=5/2=2 dir.



İlk önce sol taraf kontrol edilecek olursa 12<34<54 olduğundan değiştirme işlemi olmaz.

12	34	54	2	3

gap = 2 olduğundan ve soldaki(-) elemanla karşılaştırılma yapılacağından:
3. eleman ve (3-2). = 1. elemanlar karşılaştırılır.
2<34 olduğundan yer değiştirirler.

12	2	54	34	3

2 index öncesi array sınırlarında olmadığından bir sonraki eleman seçilir.

12	2	54	34	3

Bir üstteki işlem 3 içinde uygulanır.
3 elemanı 4.indexte bulunduğu için (4-2).=2. indexle karşılaştırılır.
4<54 olduğundan yer değiştirirler.

12	2	3	34	54

3 elemanı 2.indexte bulunduğu için (2-2).=0. indexle karşılaştırılır.
3<12 olduğundan yer değiştirirler.

3	2	12	34	54

Bu aşamadan sonra sondaki eleman en başa geldiğinden sırası ile dizinin elemanlarının ilk elemandan küçük olup olmadığına bakılır.

3	2	12	34	54

2<3 olduğundan yer değiştirme yapılır.

2	3	12	34	54

3	2	12	34	54

3	2	12	34	54

Yukarıdaki adımlar için büyük olduklarından herhangi bir işlem yapılmaz.

2	3	12	34	54

Ve sonuç olarak sıralanmış dizi elde edilir.

Insertion Sort vs. Shell Sort

Burada shell sort insertion sorta göre daha hızlı gerçekleşir. Çünkü sıralanacak olan dizi aslında alt alta ikiye bölünmüş şekilde uzayan alt dizilerin sıralanmasıyla sıralı hale gelmiş olur. Bir nevi insertion sort gibi çalışır fakat bunu sürekli diziyi ikiye bölerek gerçekleştirdiğinden daha hızlı sıralı diziyi elde eder. Daha doğrusu algoritmanın çalışmasını hızlandırır.Diğer yandan insertion sortta tek seferde sadece 1 mesafe (1 index) ötedeki elemanla swap işlemi yapılabilirken , shell sortta tek defada uzak mesafedeki bir indexteki eleman ile swap işlemi gerçekleştirilebilir. Aslında sıralanacak olan dizi zaten küçükten büyüğe doğru sıralanmışşsa insertion sort daha verimli çalışabilir(Yine insertion sort hali hazırda sıralı bir listeye eleman eklemek için de gayet verimlidir.).Ancak aksi durumda insertion sort ile baştaki bir elemanı swap işlemleriyle dizinin sonuna götürmek bir hayli zaman alacaktır. Böyle durumlarda shell sort tek seferde uzak mesafedeki indexlerle yer değiştirmeye kolaylık sağladığından daha avantajlı olacaktır sayıları taşımak için .