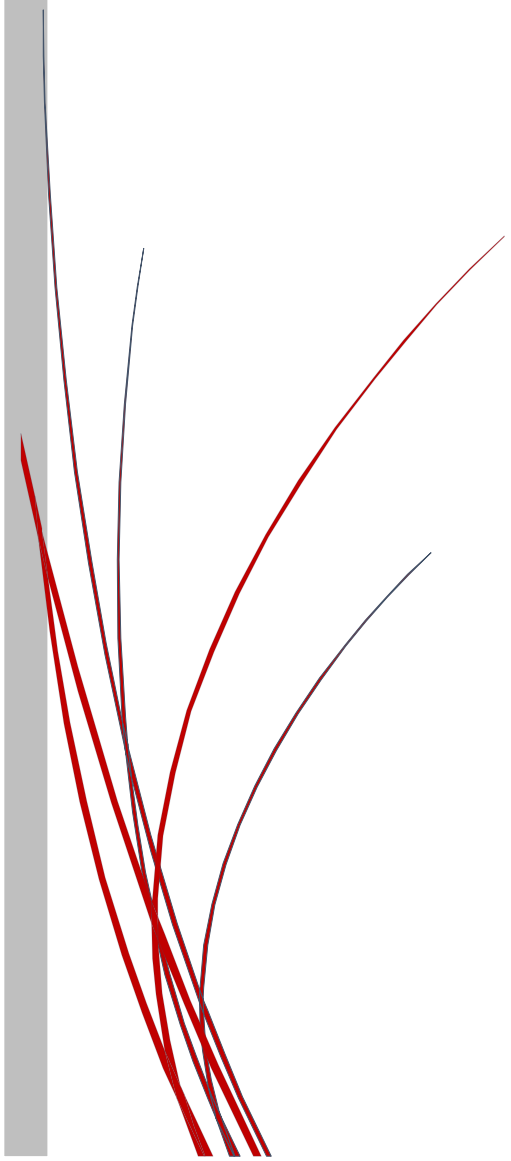
29.11.2017

CSE 321 Homework 3

Şevval MEHDER

151044009



Q1.

İmkansız olduğu söylenen bir işi yapabilmek için, öncelikle inanmak gereklidir. Tüm yetersizlik, içinde bulunulan durumun tüm çıkmazları, etrafınızdaki insanların neler söylediği sizin inancınız karşısında durabilecek kadar güçlü şeyler değildir. Başından sonuna kadar inanmanın gücünü vurgulayan, “ya yaparsak” diye yola çıkan ve bunun en azından denemek için bir fırsat olduğunu bilen insanların inatla karşı çıkışlara rağmen elde ettikleri bir başarı hikayesidir.

Mühendis olarak nasıl bir bakış açısına sahip olunması ve aynı zamanda nasıl bir bakış açısından uzak olunması gerektiğinin öğrenileceği, ekip çalışmasında esas olanın vurgulandığı, bir hedef koyup bu hedefte güzelce ilerlemenin yolunun yordamının öğretildiği yaşanmış, belgesel niteliğinde bir eser. Gerek bir işe projeye girişiken, gerek işin içerisinde zorlandığımızda, ya da moral olarak çöktüğümüzde bir desteğe ihtiyaç duyduğumuz, teknik olarak sıkıştığımızda bir mühendis olarak ne yapacağımızı bilemediğimizde, imkansızlıkla baş başa kaldığımızda ya da öyle olduğunu sandığımızda başvurabileceğimiz örnek hayatlar.

Dönemin sıkıntıları, tüm o zorluklar, daha en başından beri devamlı olan elektrik kesintileri, psikolojik baskı çok güzel yansıtılmış. Dönemin getirdiği sanayinin durumu halkın sadece daha fazla maliyet olarak gördüğü bu projenin önünde mühendisler değil, başka etkenler duruyor. Ancak şu dönemde yapabileceğimiz işlerin önünde duran sadece bizim inançsızlığımız. Ve tüm bunlar filmle beraber insana şu soruyu sordurtuyor: Bu adamlar o şartlarla bu kadar inanmış ve bunları başarmış ise, biz elimizdeki bu imkan ile başarabileceğimizin ne kadarı için çabalıyor ve ne kadarına inanıyoruz?

Q2.

Brute force bir algoritma uygulayabilmek için tüm olasılıkların denenip karar verilmesi gerekmektedir. Bu problemde, kursların sıralanması esastır. Örneğin 3x3 bir input için 3 2 1 şeklinde bir sıralama 1. Asistana 3. Kursun, 2. Asistana 2. Kursun ve 3. Asistana 1. Kursun verilmesini ifade eder. Oluşabilecek tüm bu sıralamalar permütasyon ile ifade edilebilir. Bizim problemimiz için P(kurs sayısı, asistan sayısı).

Ancak asistan sayısının kurs sayısından fazla olduğu durumlarda böyle bir permütasyon söz konusu değildir. Bu durumda, asistanın 6 saatlik iş yapacağı göz önüne alınarak kurs sayısının asistan sayısına eşitlenmesi için kare matris oluşacak şekilde 6lardan oluşan sütunlar ekledim. Örneğin aşağıdaki 3x1'lik matris şu hale geldi:

$$\begin{matrix} 3 & 5 \\ 12 & 7 \\ 3 & 5 \end{matrix} \longrightarrow \begin{matrix} 3 & 5 & 6 \\ 12 & 7 & 6 \\ 3 & 5 & 6 \end{matrix}$$

Minimum çalışma saatinin bulunması için, hesaplanan bütün permütasyon sonuçlarına göre haftalık çalışma saati toplamı elde edildi(count değişkeni). En başta en büyük integer sayının değeri ile initialize ettiğimiz optimal değişkeni ile kıyaslanarak, bu optimalden daha küçük bir sayı ise optimal count'un değerini aldı. Bu sayede optimal değişkeninde minimum çalışma saatini hesaplamış olduk.

Asistanların hangi kursu aldığı ile ilgili tek boyutlu array'in bulunması için de her adımda düşünülen permütasyon hesabına göre bir array oluşturuldu(temp değişkeni). Asistanın 6 saatlik işe atandığı durumda -1, diğer durumlarda ise o kursun sayısı eklendi. Count ve optimal kıyaslaması yapıldığı zaman eğer optimal ataması gerçekleştirilecekse return edilecek ret_asst değişkenine de bu temp değerin ataması gerçekleştirildi. Böylelikle minimum saatin bulunduğu durumda, o durumun arrayi return edilmiş oldu.

Karmaşıklığa gelindiğinde ise fonksiyonda iç içe iki adet for döngüsü bulunmaktadır. Bunlardan biri permütasyonlar kadar, diğeri ise bu permütasyonların uzunluğu yani kaçlı permütasyonu olduğu kadar dönmektedir.

Permütasyon sayısı $= P(n,r) = O(n!)$

Permütasyonların uzunluğu = asistan sayısı $= r$

Algoritmanın karışıklığı $= r.n! = O(r.n!)$

Q3.

Minimum maaliyeti hesaplayacağım bu algoritmada Breadth-First Search algoritmasından yararlandım.

1. Öncelikle verilen graph'taki vertexleri bularak başladım.

2. Bulduğum vertexlerin ilk elemanını başlangıç noktası olarak atayarak Breadth-First Search algoritması ile bağlantılı olduğu tüm komşu vertexlere ulaştım. Bu işlem sonucunda start noktasından itibaren ulaşabildiğim birbiriyle bağlantılı ziyaret edilmiş vertexlerden oluşan bir liste elde ettim.

3. Ziyaret ettiğim vertexleri, tüm vertexleri tuttuğum listeden çıkardım. Eğer listede vertex kalmışsa döngü gereği aynı işlemi ziyaret etmediğim vertexler ile yapacaktım.

4. Elde ettiğim bu listeye göre maliyet hesabı yaptım.

- Eğer lab inşa etmek, yolu tamir etmekten daha az maliyetliyse tüm vertexlere lab inşa etmek minimum maliyette olur. Döngünün bu adımındaki vertex sayısı elde ettiğim listenin (explored listesi) eleman sayısı kadardır. Eleman sayısı ile lab inşa etme maliyetini çarparsam bu adımdaki minimum maliyete ulaşmış olurum.

- Eğer lab inşa etmek daha maliyetliyse, elde ettiğim birbiriyle bağlantılı listede herhangi bir vertexe lab inşa edilir ve listedeki vertex sayısı – 1 kadar da yol tamir edilir.

Bu sayede minimum maliyet hesaplanmış olur.

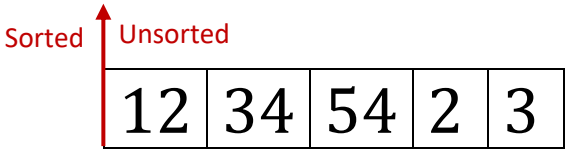
İlk adımdaki vertices listesi boş olmadığı sürece dönen while döngüsü, her vertex için çalışırsa bu durum worst case olur. Yani verilen vertexlerin hiçbir edge'i yoksa bu durum worst case olur. Ve her vertex için bfs yapılır.

İçerde yapılan Breadth-First Search için, dışardaki while döngüsü tüm vertexler için, içerdeki for döngüsü ise o vertexe ait edgeler içindir. Yani bu yapının karmaşıklığı hakkında O(V + E) denebilir. Bu da O(max(E+V)) anlamına geleceği için O(V) değerini alır.

Her vertex için O(V) karışıklığında işlem de O(V^2) karmaşıklığa sebep olur.

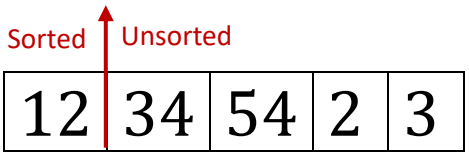
Q4. Apply Insertion sort to list 12, 34, 54, 2, 3 . Show each step in the pdf.

1



next_value = 12

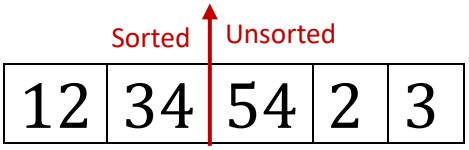
2



next_value = 34

Is 34 less than 12 ? No.

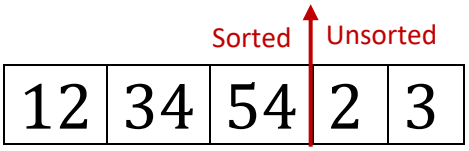
3



next_value = 54

Is 54 less than 34 ? No.

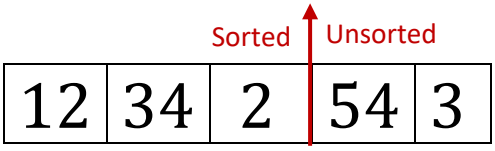
4



next_value = 2

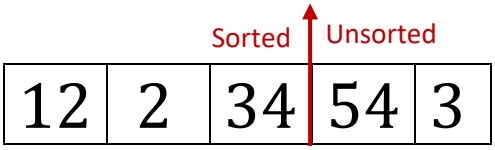
Is 2 less than 54 ? Yes, shift left.

4.1



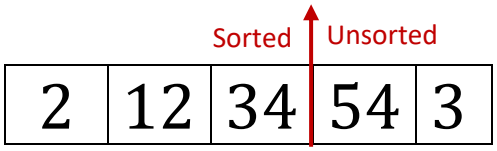
Is 2 less than 34 ? Yes, shift left.

4.2

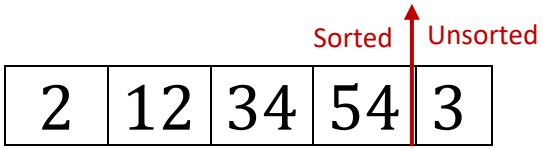


Is 2 less than 12 ? Yes, shift left.

4.3



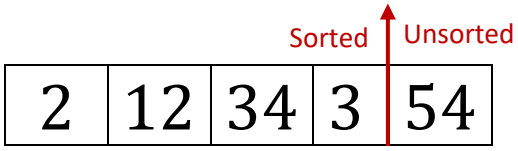
5



Next_value = 3

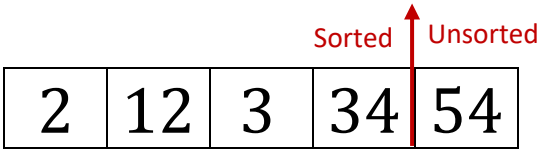
Is 3 less than 54 ? Yes, shift left.

5.1



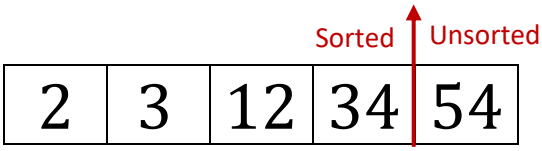
Is 3 less than 34 ? Yes, shift left.

5.2



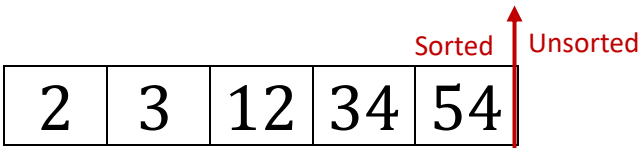
Is 3 less than 12? Yes, shift left.

5.3



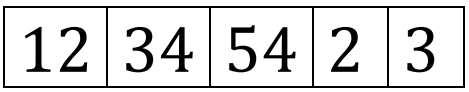
Is 3 less than 12? No.

6



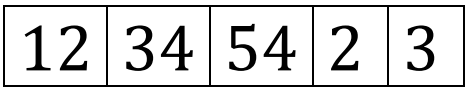
Apply **Shell sort** to list 12, 34, 54, 2, 3 . Show each step in the pdf.

1



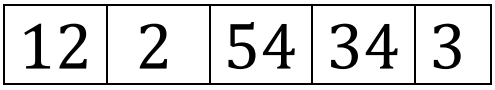
gap = 5/2 = 2
from position gap -> new_value = 54
compare with position current-gap -> 12
12 < 54

2



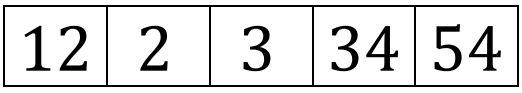
Position gap + 1 -> new_value = 2
compare with position current-gap -> 34
2 < 34

3



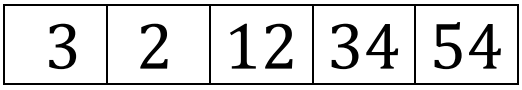
Position gap + 2 -> new_value = 3
compare with position current-gap -> 54
3 < 54

4



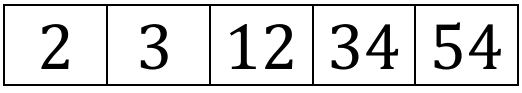
3 < 12

5



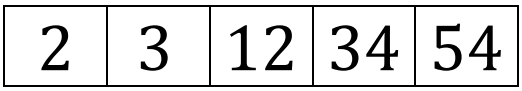
And the others already at correct position
Gap = gap / 2 = 2 / 2 = 1
position gap -> 2
compare with position current-gap -> 3
2 < 3

6



And the others already at correct position
gap = gap / 2 = 1 / 2 = 0
Sorting stops and array is sorted.

7



What are the advantages of shell sort compared to insertion sort? Discuss the differences between them.

Shell sort insertion sort'un bir çeşididir. Ancak daha iyi bir performans sağlar. Listenin bütün elemanlarını sıralamak yerine daha küçük parçaları sıralar ve her bir sorttan sonra elemanlar presorted olduğu için yapılan insertion sortun davranışı O(n^2)den çok O(n)'e yakındır. Bu sayede Shell sort daha hızlı bir sıralama sağlar. Ancak verilen listenin çok karışmamış neredeyse sıralı bir liste olduğu durumlarda insertion sort daha hızlıdır. Ancak insertion sort a göre anlaması daha zor daha karışık bir yapıya sahip. Insertion sort stabil iken Shell sort stabil değildir bu bazı durumlarda bir dezavantajdır.