GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
ALGORİTMALAR DERSİ FİNAL SINAVI

Adı – Soyadı – Numarası:

Soru 1: Aşağıda verilen fonksiyonun en kötü durum algoritma karmaşıklığı nedir? Fonksiyon farklı boyutlarda ve rastgele elemanlar ile oluşturulan diziler ile tekrar tekrar çalıştırıldığında yürütme zamanı nasıl etkilenir?

```
int ara(int[] dizi, int hedef) {
      for (int i = 0; i < dizi.length; i++) {
            if (dizi[i] == hedef) {
                  return i;
            }
      }
      return -1;
}
```

Verilen fonksiyon, dizide belirli bir hedef değeri arayan doğrusal arama (linear search) algoritmasıdır. Fonksiyonun en kötü durum karmaşıklığını belirlemek için, en uzun süren yürütme senaryosunu düşünmek gerekir. En kötü senaryo hedef değerin dizide bulunmamasıdır. Bu durumda, fonksiyon dizinin tüm elemanlarını tek tek kontrol edecektir. Dizinin boyutu arttıkça, fonksiyonun çalışma süresi de doğrusal olarak artacaktır. Bu da asimptotik olarak O(n) karmaşıklığına eşdeğerdir.

Rastgele elemanlarla oluşturulan dizilerde hedef değerin konumu değişken olacağından, fonksiyonun çalışma süresi de değişken olacaktır.

Soru 2: Aşağıda verilen fonksiyonun en kötü durum algoritma karmaşıklığı nedir? Fonksiyon farklı boyutlarda diziler ile tekrar tekrar çalıştırıldığında yürütme zamanı nasıl etkilenir?

```
void doldur(int[] dizi) {
      for (int i = 0; i < dizi.length; i++) {
            dizi[i] = random.nextInt(100);
      }
}
```

Verilen fonksiyon, bir diziyi rastgele 0 ile 99 arasındaki sayılarla doldurur. Bu fonksiyonun en kötü durumu ve ortalama durumu aynıdır çünkü her durumda dizinin tüm elemanlarına tek tek erişilir. Döngü, dizinin tüm elemanlarını baştan sona dolaşır. Bu döngünün karmaşıklığı O(n)'dir, n dizinin uzunluğudur.

Dizinin boyutu arttıkça, fonksiyonun çalışma süresi de doğrusal olarak artar. Yani, dizinin boyutu iki katına çıkarsa, fonksiyonun yürütme süresi de yaklaşık iki katına çıkar. Bu, O(n) karmaşıklığının bir sonucudur.

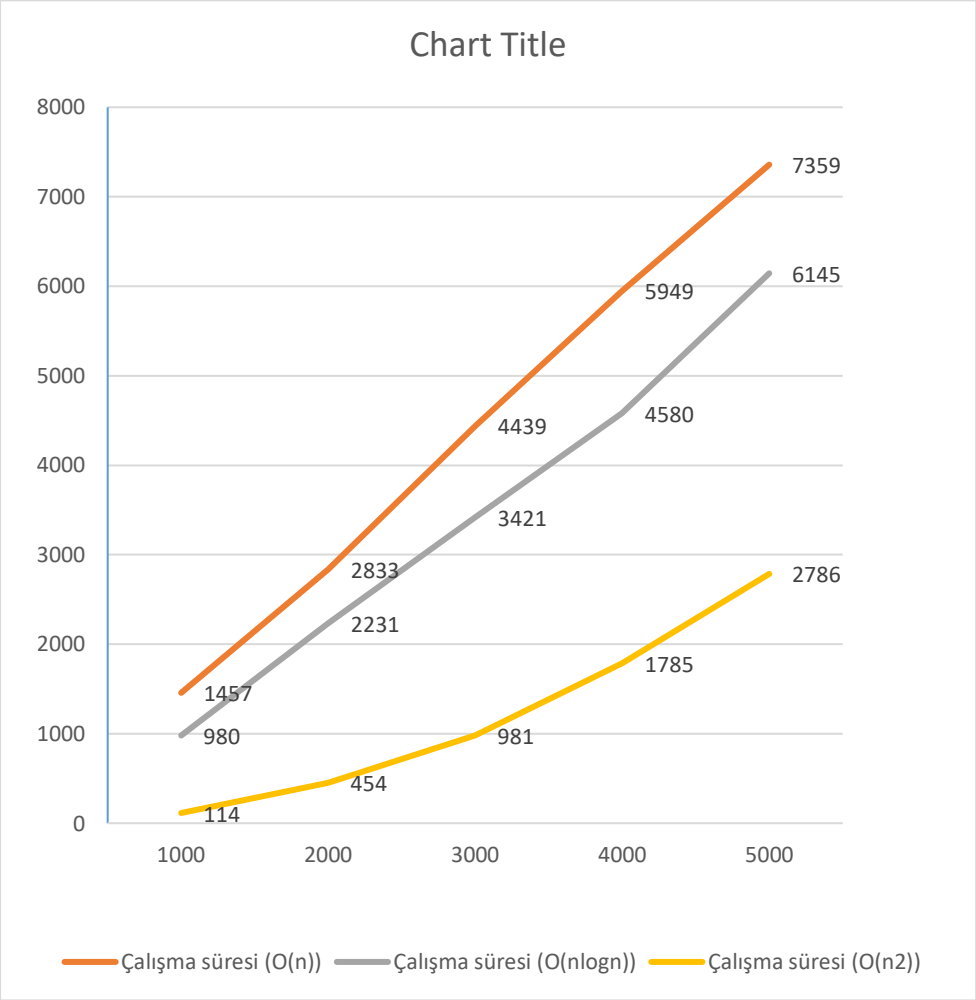
Soru 3: Aşağıda girdi boyutlarına bağlı olarak farklı algoritma karmaşıklıklarına sahip fonksiyonların yürütme süreleri verilmiştir. Boş olan hücreleri yaklaşık değerler ile doldurunuz. Grafiğini çiziniz.

Girdi boyutu	**Çalışma süresi (O(n))**	**Çalışma süresi (O(nlogn))**	**Çalışma süresi ($O(n^2)$)**
1000	1457	980	114
2000	2833	2231	454
3000	4439	3421	981

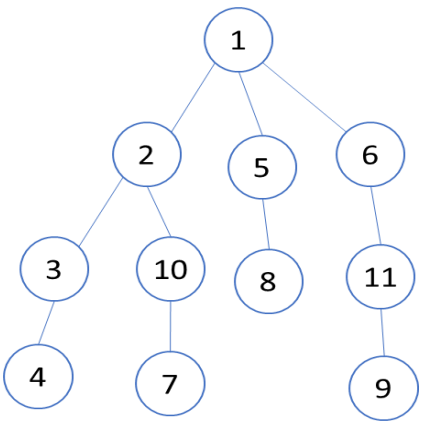
Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 27.06.2024

GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
ALGORİTMALAR DERSİ FİNAL SINAVI

4000	5949	4580	1785
5000	7359	6145	2786



Soru 4: Aşağıda verilen çizge 1 numaralı düğümden başlayarak genişlik öncelikli arama (BFS) ile gezildiğinde ziyaret edilen düğümleri sırasıyla yazınız. (gerekli ise numarası küçük olan düğüm önceliklidir)



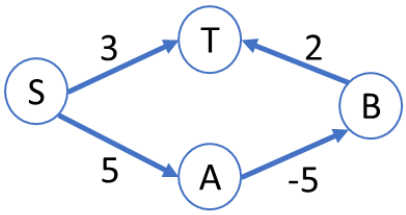
1 → 2 → 5 → 6 → 3 → 10 → 8 → 11 → 4 → 7 → 9

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 27.06.2024

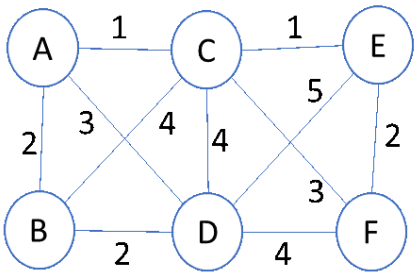
GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
ALGORİTMALAR DERSİ FİNAL SINAVI

Soru 5: Dijkstra algoritması bir kaynak düğümden diğer tüm düğümlere olan en kısa yolu bulur. Ancak, sadece pozitif ağırlıklı kenarlardan oluşan çizgelerde çalışır, negatif ağırlıklı kenarlar üzerinde çalışmaz. Nedenini kısaca açıklayınız.

Dijkstra algoritması, yalnızca pozitif ağırlıklı kenarlara sahip çizgelerde çalışır ve negatif ağırlıklı kenarlarda düzgün çalışmaz. Bunun nedeni, algoritmanın temel çalışma prensibinden kaynaklanmaktadır. Dijkstra algoritması, bir düğüme olan en kısa yolun o düğüme ilk ulaşıldığında bulunduğunu varsayar. Yani, bir düğüme ulaşıldığında, daha sonra o düğümün daha kısa bir yolla bulunması imkansızdır. Negatif ağırlıklı kenarlar bu varsayımı geçersiz kılar. Bir düğüme, negatif ağırlıklı bir kenar içeren daha kısa bir yol bulunabilir. Bu, algoritmanın daha önce bulduğu sonuçların yeniden değerlendirilmesi gerektiği anlamına gelir ki, bu Dijkstra'nın çalışma prensibine aykırıdır. Aşağıdaki örnek için; Dijkstra algoritması en kısa yol olarak S → T 'yi bulur. S → A → B → T daha kısa bir yoldur.

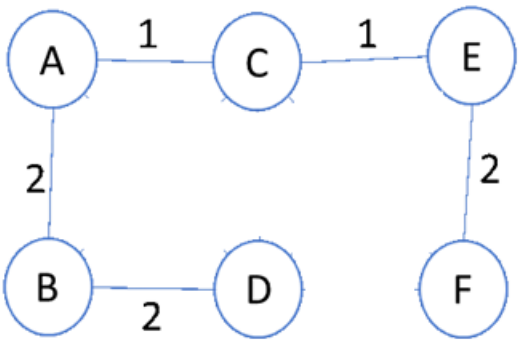


Soru 6: Kruskal algoritması bir çizgede tüm düğümleri birbirine bağlayan en kısa ağırlıklı ağacı oluşturur. Algoritmayı aşağıdaki çizge üzerinde adım adım çalıştırarak eklenen kenarları sırayla yazınız. Bulduğunuz minimum kapsayan ağacı (MST) çiziniz.



Öncelikle kenarlar ağırlıklarına göre küçükten büyüğe sıralanır. Aynı ağırlığa sahip kenarlar olduğu için her adımda eklenen kenarların sırası değişiklik gösterebilir. Bu örnek için sonuçta oluşan MST değişmeyecektir.

A – C, C – E, A – B, E – F, B – D, D – F



Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 27.06.2024

Soru 7: Herhangi bir çizgede tüm kenarların ağırlıkları 1 arttırılırsa minimum kapsayan ağacı değişir mi? Açıklayınız.

Herhangi bir çizgede tüm kenarların ağırlıkları 1 arttırılırsa minimum kapsayan ağaç (MST) değişmez. Çizgedeki tüm kenar ağırlıklarının 1 artırılması, her kenar ağırlığının aynı sabit miktarda artması anlamına gelir. Örneğin, başlangıçta bir kenarın ağırlığı w ise, ağırlığı w+1 olacaktır. MST'yi oluştururken önemli olan, iki düğüm arasındaki nispi ağırlık farklarıdır, yani hangi kenarın diğerlerinden daha hafif olduğu önemlidir. Tüm kenar ağırlıkları aynı miktarda artırıldığında, kenarların nispi ağırlık farkları değişmez.

Soru 8: Kruskal algoritmasının karmaşıklığı O(ElogE)'dir. Algoritmanın hangi adımı bu duruma sebep olur?

Kruskal algoritmasının zaman karmaşıklığının O(ElogE) olmasının nedeni, algoritmanın kenarları ağırlıklarına göre sıralama adımıdır. Kullanılan sıralama algoritmasına göre Kruskal algoritmasının zaman karmaşıklığı değişir. Kenarlar sıralandıktan sonra, kenarlar teker teker işlenerek döngü oluşturup oluşturmadıkları kontrol edilir. Bu adımda kullanılan Birleştir-Bul (Union-Find) veri yapısı, path compression (yol sıkıştırma) ve union by rank (seviyeye göre birleştirme) gibi teknikler kullanılarak optimize edildiğinde, her bir işlem neredeyse sabit zaman alır, amortize edilmiş zaman karmaşıklığı O(α(n)) olur. Burada α, çok yavaş büyüyen ters Ackermann fonksiyonudur ve pratikte sabit olarak kabul edilir.

Soru 9: Minimum kapsayan ağaç yerine maksimum kapsayan ağacı bulmak için Kruskal algoritması kullanılabilir mi? Nasıl bir değişiklik yapmak gerekir?

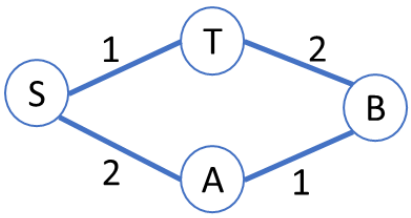
Kruskal algoritması maksimum kapsayan ağacı (MST) bulmak için de kullanılabilir. Standart Kruskal algoritmasında kenarlar ağırlıklarına göre artan sırada sıralanır. Maksimum kapsayan ağacı bulmak için kenarlar ağırlıklarına göre azalan sırada sıralanır.

Soru 10: Bir çizgenin birden fazla minimum kapsayan ağacı olabilir mi? Açıklayınız.

Evet, bir çizgenin birden fazla minimum kapsayan ağacı (MST) olabilir. Eğer çizgede birden fazla kenar aynı ağırlığa sahipse, bu kenarlar arasında seçim yaparken farklı kombinasyonlar mümkün olabilir ve bu da farklı MST'lerin oluşmasına yol açabilir. Aşağıdaki örnek için;

MST 1: S – T, A – B, T – B

MST 2: S – T, A – B, S - A



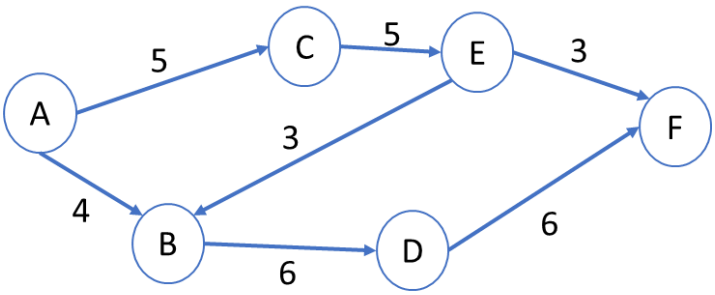
Soru 11: Maksimum akış bulma probleminde artık yolları (augmenting paths) bulmak için Ford-Fulkerson algoritması DFS, Edmonds-Karp algoritması BFS yaklaşımını kullanır. Bu iki farklı yaklaşım nasıl bir sonuca neden olur?

Maksimum akış problemini çözmek için kullanılan Ford-Fulkerson ve Edmonds-Karp algoritmaları, artık yolları (augmenting paths) bulmak için farklı arama stratejileri kullanır: Ford-Fulkerson derinlik öncelikli

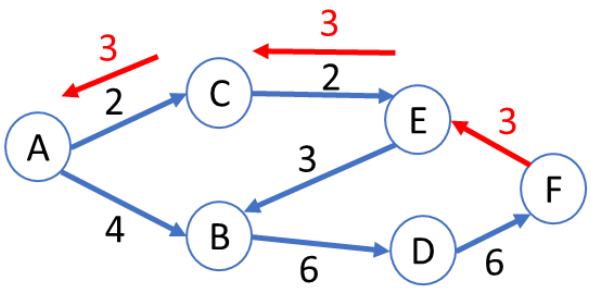
Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 27.06.2024

arama (DFS) kullanırken, Edmonds-Karp genişlik öncelikli arama (BFS) kullanır. DFS, rastgele ve derinlemesine seçimler yaparak artık yolları bulur. Bu nedenle, seçilen yollar her zaman en kısa veya en etkili yollar olmayabilir. Bu, algoritmanın daha fazla adımda sonuca ulaşmasına neden olabilir. BFS, kaynak düğümden başlayarak en kısa artık yolu (en az sayıda kenar içeren) bulur. BFS ile en kısa yollar bulunarak yapılan artırmalar, genel olarak daha az adımda maksimum akışa ulaşmayı sağlar.

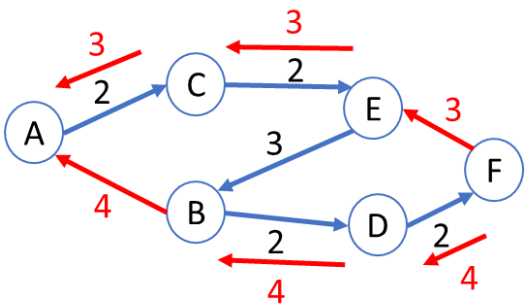
Soru 12: Aşağıda verilen çizgede seçtiğiniz bir algoritmaya göre maksimum akışı bulunuz. Her bir adımda bulduğunuz yolu ve kapasitesini yazınız ve çizgenin güncel durumunu çiziniz.



A → C → E → F kapasite 3

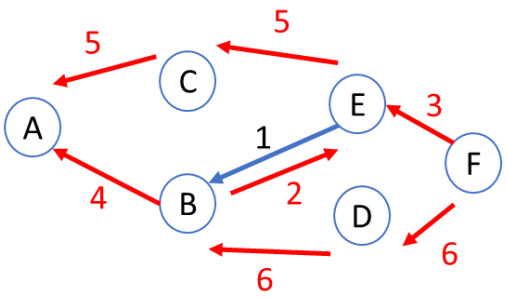


A → B → D → F kapasite 4



A → C → E → B → D → F kapasite 2

GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
ALGORİTMALAR DERSİ FİNAL SINAVI



Soru 13: Rabin-Karp algoritması dizge eşleme (string matching) sırasında hash fonksiyonu kullanır. Örneğin; "abracadabra" metni içerisinde "cad" örüntüsünü ararken, örüntünün hash değerini hesaplar. Metindeki her olası alt dizgenin hash değerlerini hesaplar ve bu hash değerlerini karşılaştırarak arama yapar. Metin boyunca kayan bir pencere yöntemi kullanılarak (her defasında bir karakter kaydırılarak) her adımda metnin o andaki alt dizgesinin hash değeri hesaplanır ve bu değer, örüntünün hash değeri ile karşılaştırılır. Hash değerlerini karşılaştırmanın dezavantajı nedir?

Hash çakışması, iki farklı dizgenin aynı hash değerine sahip olması durumudur. Bu, algoritmanın iki farklı dizgeyi aynıymış gibi değerlendirmesine yol açar. Çakışma durumunda, hash değerleri aynı olan dizgeler için tam karakter karşılaştırması yaparak gerçek eşleşmenin doğrulanması gerekir. Çakışma olasılığını azaltmak için daha güçlü ve daha az çakışma üreten hash fonksiyonları kullanılabilir. Hash değerlerinin hesaplanmasında büyük asal sayılar kullanılarak modüler aritmetik uygulanır. Algoritma her potansiyel eşleşmede tam karakter karşılaştırması yaparak çakışmaları önleyebilir. Ancak, bu ek işlemler maliyetlidir.

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 27.06.2024