GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
ALGORİTMALAR DERSİ BÜTÜNLEME SINAVI

**Adı – Soyadı – Numarası:**

**Soru 1:** Aşağıda verilen fonksiyonun en kötü durum algoritma karmaşıklığı nedir? 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 - 1; i++) {
            if (dizi[i] > dizi[i+1]) {
                  return 0;
            }
      }
      return -1;
}
```

Bu fonksiyon, verilen bir dizinin sıralı olup olmadığını kontrol eder. Dizi elemanları arasında bir düzensizlik (yani bir elemanın kendisinden sonraki elemandan büyük olması) varsa 0, aksi halde -1 döndürür.

Döngü, dizinin n elemanı varsa n-1 kez çalışır. Her döngü adımında bir karşılaştırma yapılır. Bu nedenle, döngü her durumda n-1 kez çalışır ve her bir döngü adımında bir karşılaştırma yapılır. En kötü durumda, dizi tamamen sıralıdır ve döngü sonuna kadar çalışır. Bu durumda, döngü n-1 kez çalışır. Dolayısıyla, en kötü durum algoritma karmaşıklığı O(n) olarak ifade edilir.

Fonksiyon farklı boyutlarda ve rastgele elemanlar ile oluşturulan dizilerle tekrar tekrar çalıştırıldığında, yürütme zamanı dizinin boyutuna ve dizideki elemanların sıralı olup olmamasına bağlı olarak değişir: Eğer dizi tamamen sıralıysa, döngü dizinin sonuna kadar çalışır ve yürütme zamanı O(n) olur. Eğer dizi düzensizse, düzensizliğin ilk bulunduğu anda döngü kırılır ve fonksiyon 0 döner. Bu durumda yürütme zamanı O(k) olur, burada k düzensizliğin bulunduğu indekstir ve k en fazla n-1 olabilir.

Özetle, fonksiyonun en kötü durum algoritma karmaşıklığı O(n)'dir ve yürütme zamanı dizinin boyutuna ve dizinin sıralı olup olmamasına bağlı olarak değişir. Diziler rastgele oluşturulduğunda, düzensizliğin erken bulunma olasılığı olduğundan, ortalama yürütme zamanı en kötü durumdan daha iyi olacaktır.

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

```
int doldur(int[] dizi) {
      int toplam = 0;
      for (int i = 0; i < dizi.length; i++) {
            toplam += dizi[i];
      }
      return toplam;
}
```

Bu fonksiyon, verilen dizideki tüm elemanların toplamını hesaplar. Döngü, dizinin n elemanı varsa n kez çalışır. Her döngü adımında bir toplama işlemi yapılır.

Döngü, dizinin her elemanı için bir kez çalıştığından, n elemanlı dizi için döngü n kez çalışır. Dolayısıyla, en kötü durum algoritma karmaşıklığı O(n) olarak ifade edilir.

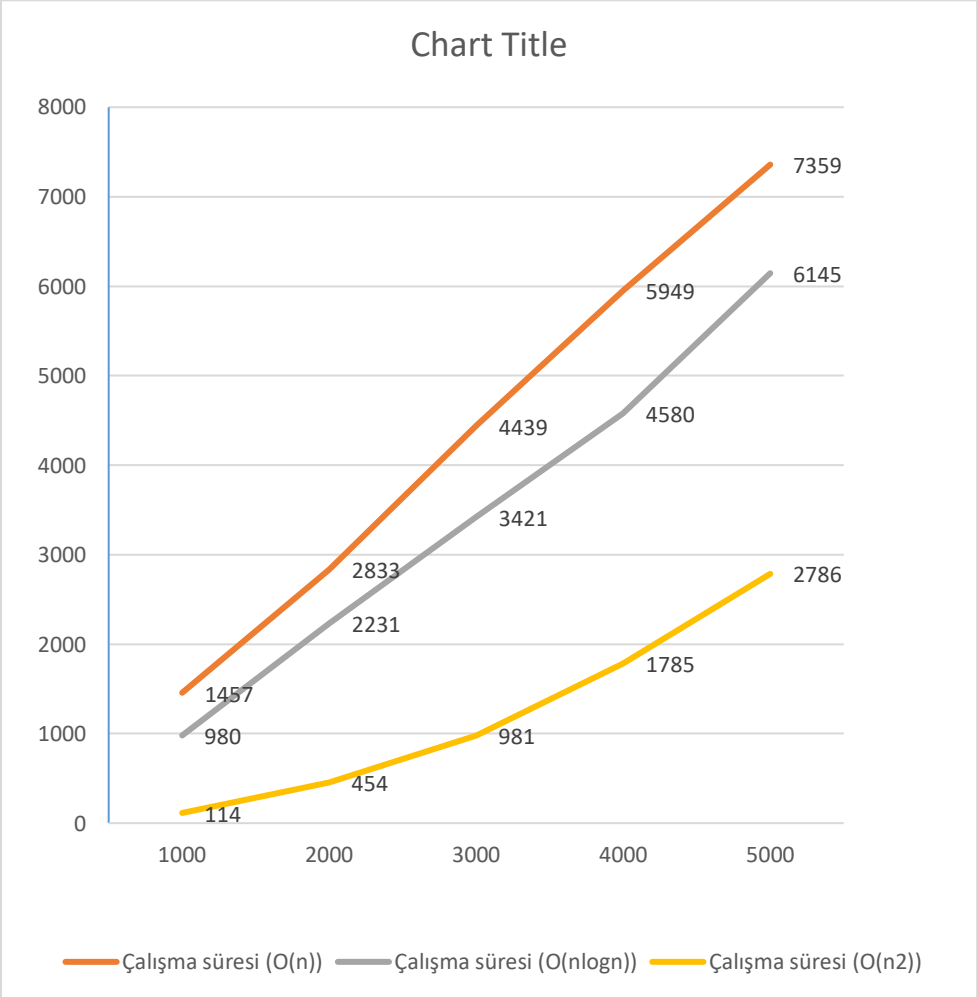
Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 12.07.2024

GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
ALGORİTMALAR DERSİ BÜTÜNLEME SINAVI

Fonksiyon farklı boyutlarda diziler ile tekrar tekrar çalıştırıldığında, yürütme zamanı doğrudan dizinin boyutuna bağlıdır: Dizi boyutu arttıkça, fonksiyonun çalışması gereken döngü sayısı da n artar. Bu nedenle, dizinin boyutu arttıkça fonksiyonun yürütme zamanı doğrusal olarak artar. Fonksiyon, dizinin elemanlarının değerlerine bağımlı değildir, sadece eleman sayısına bağlıdır. Bu nedenle, dizideki elemanların değerleri yürütme zamanını etkilemez.

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

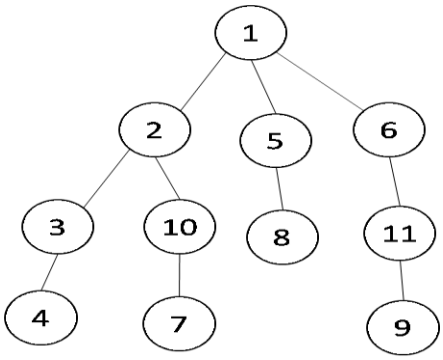
| **Girdi boyutu** | **Çalışma süresi (O(n))** | **Çalışma süresi (O(nlogn))** | **Çalışma süresi (O($n^2$))** |
|---|---|---|---|
| 1000 | 1457 | 980 | 114 |
| 2000 | 2833 | 2231 | 454 |
| 3000 | 4439 | 3421 | 981 |
| 4000 | 5949 | 4580 | 1785 |
| 5000 | 7359 | 6145 | 2786 |




Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 12.07.2024

GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
ALGORİTMALAR DERSİ BÜTÜNLEME SINAVI

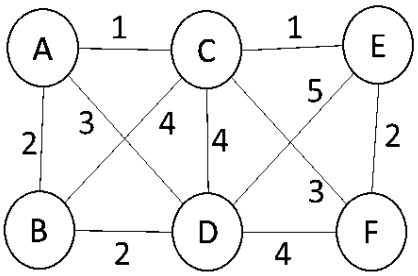
**Soru 4:** Aşağıdaki çizge 1 numaralı düğümden başlayarak genişlik öncelikli arama (BFS) ve derinlik öncelikli arama (DFS) ile gezildiğinde ziyaret edilen düğümleri sırasıyla yazınız. (numarası küçük düğüm öncelikli)




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

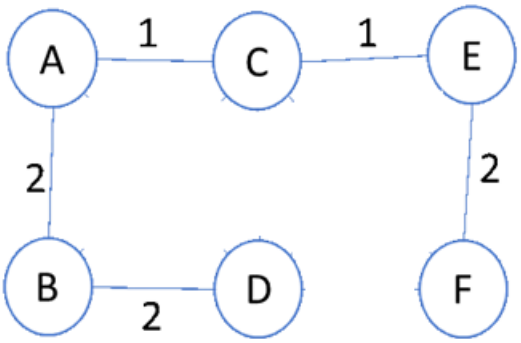
DFS: 1 → 2 → 3 → 4 → 10 → 7 → 5 → 8 → 6 → 11 → 9

**Soru 5:** Prim ve Kruskal algoritması bir çizgede tüm düğümleri birbirine bağlayan en kısa ağırlıklı ağacı oluşturur. İki algoritmayı aşağıdaki çizge üzerinde adım adım çalıştırarak MST'ye eklenen kenarları sırayla yazınız. Bulduğunuz minimum kapsayan ağacı (MST) çiziniz.




Kruskal: 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




Prim: Aynı ağırlığa sahip kenarlar olduğunda kenarların MST'ye eklenme sıraları değişebilir. Bu örnek için sonuçta oluşan MST değişmeyecektir.

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 12.07.2024

Kruskal algoritmasından farklı olarak, A – C kenarı eklenmeden C – E kenarı eklenemez. C – E kenarı eklenmeden E – F kenarı eklenemez. A – B kenarı eklenmeden B – D kenarı eklenemez.

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

**Soru 6:** 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 7:** Minimum kapsayan ağaç yerine maksimum kapsayan ağacı bulmak için Kruskal algoritması kullanılabilir mi? Nasıl bir değişiklik yapmak gerekir?

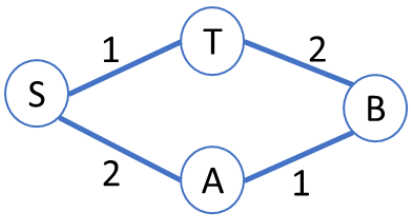
Kruskal algoritması maksimum kapsayan ağacı (MST) bulmak için de kullanılabilir. Standart Kruskal algoritmasında kenarlar ağırlıklarına göre artan sırada sıralanır. Maksimum kapsayan ağacı bulmak için kenarlar ağırlıklarına göre azalan sırada sıralanır.

**Soru 8:** 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 9:** 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?

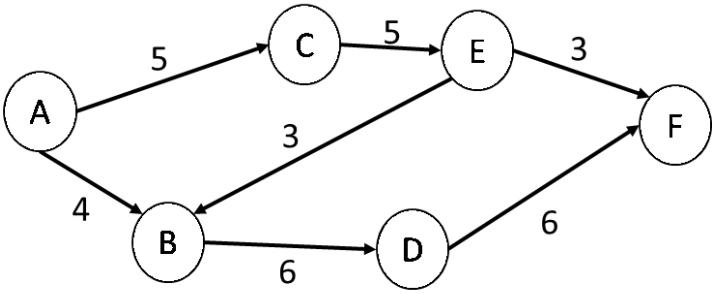
Ford-Fulkerson algoritması, DFS kullanarak her seferinde bir artık yol bulur ve bu yolu boyunca mümkün olan maksimum akışı arttırır. En kötü durumda, her DFS çağrısı, ağdaki tüm kenarları kontrol edebilir, bu nedenle bir artık yol bulmanın karmaşıklığı O(E) olabilir. Her DFS çağrısı, mevcut kapasiteye bağlı olarak küçük artışlarla akışı arttırabilir. Bu nedenle, en kötü durumda, toplam çalışma zamanı O(Ef) olabilir, burada E kenar sayısı ve f maksimum akıştır. DFS, genellikle daha uzun yolları bulmaya yatkındır, ve bu yolların kapasitesi küçük olabilir. DFS kullanımı, belirli çizgelerde performansın öngörülemez ve verimsiz olmasına neden olabilir.

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 12.07.2024

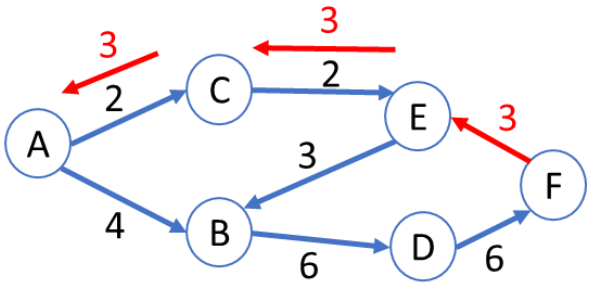
GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
ALGORİTMALAR DERSİ BÜTÜNLEME SINAVI

Edmonds-Karp algoritması, BFS kullanarak her seferinde en kısa (kenar sayısı açısından) artık yolu bulur. BFS'nin bir artık yolu bulma karmaşıklığı O(V+E)'dir, burada V düğüm sayısı ve E kenar sayısıdır. BFS her seferinde en kısa yolu bulduğu için, her artış en azından bir kenar boyunca yapılır. Bu durumda, maksimum akış değeri f olduğunda, her artış O(VE) zaman alabilir. Toplamda, Edmonds-Karp algoritmasının çalışma zamanı $O(VE^2)$ olur. BFS, her zaman en kısa artık yolu bulur, bu da algoritmanın düzenli ve öngörülebilir çalışmasına neden olur. BFS kullanımı, daha kısa yollar bulduğu için daha az sayıda yineleme gerektirir ve belirli çizgelerde daha verimlidir.

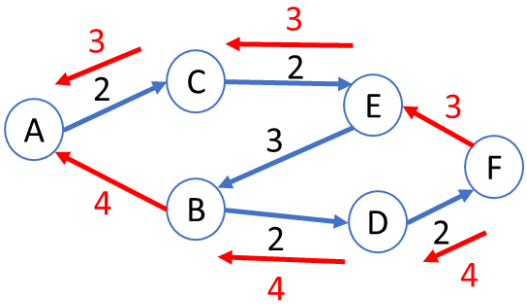
**Soru 10:** Aşağıda verilen çizgede seçtiğiniz bir algoritmaya göre maksimum akışı bulunuz. Her bir adımda bulduğunuz yolu ve kapasitesini yazınız ve çizgenin güncel durumunu çiziniz.




A → C → E → F kapasite 3



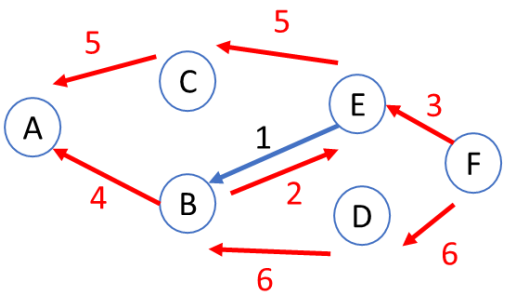

A → B → D → F kapasite 4




A → C → E → B → D → F kapasite 2

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 12.07.2024

GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
ALGORİTMALAR DERSİ BÜTÜNLEME SINAVI




Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 12.07.2024