T.C. Fırat Üniversitesi
Bilgisayar Mühendisliği Bölümü

BMÜ316-Algoritma Analizi Dersi

3. Ödev Raporu

Ders Sorumlusu: Prof. Dr. Mehmet KARAKÖSE

Öğrenci İsim: Mert İNCİDELEN
Öğrenci No.: 170260101

Ödev Tanımı

BMÜ316 - Algoritma Analizi Dersi 3. Ödevinde,
1. En az 3 tane farklı NP probleme karşılık gerçek dünya problemlerinin çözümleri için algoritmaların ve karmaşıklıklarının ve
2. En az 3 farklı ağaç yapısının gerçek dünya problemlerinin çözümünde kullanıldığı örneklerin, algoritmaları ile karmaşıklıklarının verilmesi beklenmektedir.

1. NP Problemlere Karşılık Gelen Gerçek Dünya Problemleri, Çözüm Algoritmaları ve Karmaşıklıkları

1.a. Traveling Salesman Problem (Gezgin Satıcı Problemi)

Gezgin satıcı problemi günlük yaşamda bilgisayar bilimi, genetik ve mühendislik, lojistik gibi birçok alanda karşılık bulmaktadır. Örneğin baskılı devre kartlarının delme işleminde, deliklerin boyutlarının farklı olma durumundan dolayı makinenin farklı delme ekipmanını kullanması gerekir. Burada makine hareket süresini en aza indirmek için belirli çaptaki tüm deliklerin delinmesi daha sonra başka belirli bir çap seçerek diğer deliklerin oluşturulması gerekir. Burada bir gezgin satıcı problemi ortaya çıkmaktadır.

Günlük Yaşam Örneği: "Tedarikçi Teslimat Plânı"

Ele aldığım ürün tedarik sürecindeki Araç Yönlendirme Sorunu (VRP) Gezgin Satıcı Probleminin bir genel halidir. Bir tedarikçinin, müşterilerine depodan çeşitli malları ulaştırma durumunda kamyonlara bir müşteri ataması ve her bir kamyon için bir teslimat programı bulması ve böylece her bir kamyonun kapasitesinin aşılmadan ve toplam seyahat mesafesinin en aza indirmesi gerekir. Belirli bir bölge için teslimat planı için en kısa mesafenin tespitinde kaba kuvvet yaklaşımıyla aşağıdaki algoritmanın aşamaları takip edilebilir.

ADIM 1: Her kenardan bir kez geçmek kaydıyla tüm düğümlerin ziyaret edilmesiyle bir yol bulunarak, bulunan yolun maliyeti en iyi yol maliyeti olarak alınır.
ADIM 2: Teslimat noktaları için her kenardan bir kez geçmek kaydıyla tüm düğümler ziyaret edilerek yeni bir yol saptanır.
ADIM 3: Yeni saptanan yolun maliyeti şimdiye kadarki en iyi yol maliyetinden düşükse, en iyi yol maliyeti olarak alınır.
ADIM 4: Tüm yollar denenmediyse ADIM 2 yeniden uygulanır, denendiyse bulunan en iyi yol maliyeti en kısa mesafedir.

Zaman Karmaşıklığı: O(n!)

Burada tüm teslimat noktalarının ulaşım yollarının kombinasyonları oluşturulur. Dolayısıyla (n-1)! adet yol bulunması gerekir. Bu nedenle kaba kuvvet yaklaşımıyla gerçekleştirilen bu çözümün zaman karmaşıklığı, n teslimat noktası sayısı olmak üzere **O(n!)**'dir. Teslimat noktası sayısının yüksek sayılara ulaşması, kombinasyonların sayısının astronomik boyutlarda artması anlamına gelecektir.

1.b. Knapsack Problem (Sırt Çantası Problemi)

Sırt çantası problemi günlük yaşamda sınırlı kapasiteye sahip bir durum için maksimize edilmiş kazanca yönelik gerekli kaynakları tahsis edilmesi gereken durumlarda kullanılır. Örneğin bir doğa gezgininin çantasına koyacaklarının seçimi, bir kargo uçağının yüklenmesi gibi sorunlar sırt çantası problemi çerçevesinde değerlendirilebilir.

Günlük Yaşam Örneği: "Lojistik Konteyner Doldurma Sorunu"

Knapsack Problem için lojistik konteyner doldurma sorununu ele alırsak, konteyner kapasitesine göre yüksek kazanç elde edilecek malzemelerin seçilmesi ve yüklenmesi gerekmektedir. Bu durumda ürünlerin boyutları ve ürünlerden kazanç miktarları farklıdır. Konteyner kapasitesine göre en iyi sonucu alacak şekilde ürünlerin yüklenmesi ve kazancın maksimize edilmesi gerekmektedir. Knapsack problemi çözümünde takip edilecek algoritma için toplam kazancın hesaplandığı bir sözde kod şöyle olacaktır:

```
toplamKazanc(kapasite, urunBoyut, urunKazanc, n)

    if n == 0 or kapasite == 0 then
        return 0
    end if

    if urunBoyut[n-1] > kapasite then
        return toplamKazanc(kapasite, urunBoyut, urunKazanc,
    n-1)
    else
        return max(kazanc[n-1] + toplamKazanc(kapasite-
    urunBoyut[n-1], boyut, urunKazanc, n-1), kazanc(kapasite,
    urunBoyut, urunKazanc, n-1))
    end if
```

Zaman Karmaşıklığı: $O(2^n)$

Her aşamada problemler tekrar hesaplanacağından ve fonksiyon çağırılacağından özyinelemeli bu algoritmanın karmaşıklığı **$O(2^n)$**'dir. Ürün miktarı arttıkça hesaplama süresi 2 tabanında üssel artacaktır.

1.c. Bin Packing Problem (Kutu Paketleme Problemi)

Kutu paketleme problemi gerçek dünyada farklı boyutların sabit boyutlara sığdırılması işlemleri için kullanılabilir. Örneğin kamyonların yüklenmesi, verilerin sabit uzunluklu disklerde depolanması gibi sorunlar için kutu paketleme probleminden söz edilebilir. Aşağıdaki örnekte TV kanallarının sabit süreli reklam kuşaklarına farklı sürelerde reklamları sığdırması ele alınmıştır.

Günlük Yaşam Örneği: "TV Reklamlarını, Reklam Aralarına Sığdırma"

Televizyon kanallarının reklam aralıkları belirli sürelerle sabittir. Bu sabit uzunluktaki reklam kuşaklarına farklı sürelere sahip birden çok reklamın optimum olarak dağıtılması gerekmektedir.

Farklı reklam sürelerinin sabit bir reklam kuşağı süresine sığdırılması ile gereken reklam kuşağı sayısını bulmak için gereken algoritmanın sözde kodu aşağıdaki gibidir. Hesaplama için Next Fit Algoritması kullanılmıştır. Algoritmada, bir sonraki süre işlenirken son süreyle aynı reklam kuşağına sığıp sığmadığı kontrol edilir, sığmıyorsa yeni reklam kuşağı kullanılır.

```
reklamKusagiSayisi(reklamlar[], reklamSayisi, reklamKusagi)
    sonuc = 0
    kalanSure = reklamKusagi

    for i=0, 1, 2, ... , reklamSayisi do

        if reklamlar[i] > kalanSure then
            sonuc++
            kalanSure = reklamKusagi – reklamlar[i]
        else
            kalanSure = kalanSure – reklamlar[i]
        end if

    end for

    return sonuc
```

Zaman Karmaşıklığı: O(n)

Next Fit Algoritması çözümü basit bir çözümdür. Burada n adet reklam süresi kontrol edileceğinden zaman karmaşıklığı **O(n)** olmaktadır.

2. Ağaç Yapılarının Gerçek Dünya Problemlerinin Çözümünde Kullanılması ve Algoritmaların Karmaşıklıkları

2.a. Heap Tree (Yığın Ağacı)

Heap ağaç yapısı, gerçek dünyada öncelik gerektiren durumların tespiti için kullanılabilmektedir. Örneğin bir banka sıra makinesinde sıra için kayıt yapıldıktan sonra müşteriler, randevulu olma durumları ve müşteri statüleri gibi farklı öncelik derecelerine göre çağırılabilmektedirler. Aşağıda ele aldığım problemde ise hastanede kaydı yapılan hastaların öncelik durumlarına göre çağırılabilmesi örneklenmiştir.

Günlük Yaşam Kullanımı Örneği: “Randevu Sistemi”

Bir poliklinikte bekleyecek olan hastaların yaşlı olma durumu, engelli olma durumu, hamile olma durumu, randevulu olma durumu, hastalığın acil olma durumu gibi birbirilerinden farklı öncelik dereceleri bulunabilmektedir. Buna göre tasarlanan sistem tarafından, hastaların bu önceliklere göre çağırılması beklenir. Örnekte öncelik belirlemesi için maksimum öncelikli heap ağaç yapısı kullanılmıştır.

Hastanın Kaydının Alınması (Maksimum Öncelikli Heap Ağacına Ekleme) İşlemi:

ADIM 1: Hastanın kaydı için bir düğüm oluşturulur.
ADIM 2: Hasta durumunu karşılaştırarak bir sayısal öncelik değeri belirlenir. (Örneğin hamile hasta için belirlenen değerin, öncelik durumları dışında kalan bir hasta için belirlenen değerden büyük olması gerekir.)
ADIM 3: Düğüm, ağacın en alt ve en sağ boş alana (n+1. eleman olarak) eklenir.
ADIM 4: Düğümün değeri kendi ebeveyni ile karşılaştırılır. Düğüm değeri ebeveyn değerinde büyükse ADIM 5 uygulanır, değilse ADIM 6 uygulanır.
ADIM 5: Düğüm ile ebeveyn düğüm yer değiştirilir. ADIM 4 yeniden tekrarlanır.
ADIM 6: Ekleme işlemi sonlandırılır.

Heap isimli bir düğüm listesi için n, heap'in son eleman indisi olmak üzere algoritmanın sözde kodu şöyle olur:

```
ekleme(hasta, n)

    heap[n+1] = hasta
    dugum = n+1

    while heap[dugum] > heap[dugum/2] do

        temp = heap[dugum]
        heap[dugum/2] = heap[dugum]
        heap[dugum] = temp

        dugum = dugum/2

    end while
```

Zaman Karmaşıklığı: O(logn)

Ekleme işleminde, eklenen düğüm her aşamada kendi kökü ile karşılaştırılacağından maksimum ağacın derinliği kadar karşılaştırma yapılır. Dolayısıyla heap ağacında bulunan eleman sayısı n ise, ekleme işlemi **O(log n)** zamanda tamamlanır.

Sıradaki Hastanın Çağırılması (Heap Ağacının Kökünü Silme ve Düzeltme) İşlemi:

ADIM 1: Sırası gelen hastanın bulunduğu kök düğüm çıkarılır.
ADIM 2: Çıkarılan kök düğüm yerine ağacın en alt ve en sağ düğümü (n. elemanı) eklenir.
ADIM 3: Düğüm, çocuk düğümleri ile karşılaştırılır. Ebeveyn düğüm çocuk düğümlerden herhangi birinden küçük olma durumunda ADIM 4 uygulanır, değilse ADIM 5 uygulanır.
ADIM 4: Düğüm ile çocuk düğüm yer değiştirilir. ADIM 3 yeniden tekrarlanır.
ADIM 5: Silme işlemi sonlandırılır.

Heap isimli bir düğüm listesi için silme ve düzeltme algoritmasının sözde kodu şöyle olur:

```
hastaCagir()
        hasta = heap[1]
        heap[1] = heap[sonEleman]
        düzelt(1)
        return hasta

duzelt(n)
        if heap[n] < heap[2*n] or heap[n] < heap[(2*n)+1] then
                if heap[2*n] > heap[2*n+1] then
                        temp = heap[n]
                        heap[n] = heap[2*n]
                        heap[2*n] = temp
                        duzelt(2*n)
                else
                        temp = heap[n]
                        heap[n] = heap[2*n+1]
                        heap[2*n+1] = temp
                        duzelt(2*n+1)
                end if
        end if
```

Zaman Karmaşıklığı: O(logn)

Silme işleminde, kök düğüm yerine n. eleman alındıktan sonra ağacın düzeltilmesi için her aşamada kök düğüm çocuk düğümlerle karşılaştırılır. Düğüm en fazla ağacın derinliği kadar çocuk düğümlerle karşılaştırılacağından silme işlemi **O(log n)** zamanda tamamlanır.

2.b. Minimum Spanning Tree (Asgari Tarama Ağacı)

Asgari tarama ağacı ve algoritmaları günlük yaşamda telekomünikasyon ağları, ulaşım ağları, su şebekeleri ve elektrik şebekelerinin en az maliyetle oluşturulabilmesi için kullanılabilmektedir.

Günlük Yaşam Kullanımı Örneği: "Kablo Hattı Oluşturma"

Aşağıda ele aldığım problemde şehirler arası bir telefon kablosu döşeme işleminde asgari tarama ağacının oluşturulma işlemi kruskal ve prim algoritması kullanılarak örneklenmiştir. Bölgeler (düğümler) arası birçok alternatif yol bulunabilmektedir.

Prim Algoritması İle Minimum Tarama Ağacının Oluşturulması:

Prim algoritması minimum tarama ağacı oluştururken, herhangi bir düğümden başlar ve halka oluşturmadan diğer düğümlere uğrar.

ADIM 1: Bir şehir (düğüm) seçilerek minimum tarama ağacı oluşturmaya başlanır.
ADIM 2: Ağacı yeni düğümlere bağlayan yollar seçilir. Bu yollar arasında en kısa yol bulunur.
ADIM 3: Bu yol üzerinden bir döngü oluşuyorsa yol seçenekler arasından çıkarılır ve ADIM 2 yeniden uygulanır, oluşmuyorsa ADIM 4 uygulanır.
ADIM 4: Seçilen yol işaretlenerek düğüm mevcut ağaca eklenir. Ağaca eklenmeyen düğüm var ise ADIM 2 uygulanır, yok ise ADIM 5 uygulanır.
ADIM 5: Tüm düğümler ağaca eklendiğinden asgari tarama ağacı oluşturma işlemi tamamlanır.

Zaman Karmaşıklığı: $O(n^2)$

n düğüm için her aşamada n düğüme bir kez bakmak durumunda olduğu için karmaşıklık **$O(n^2)$**'dir.

Kuruskal Algoritması İle Minimum Tarama Ağacının Oluşturulması:

Kruskal algoritması minimum tarama ağacı oluştururken düşük mesafeden yüksek mesafeye kadar tüm kenarları sıralı olarak değerlendirir ve kapalı bir döngü oluşturmayacak şekilde kenarları tarar ve ekler.

ADIM1: Sıralı kenar listesinden sıradaki kenar alınır.
ADIM2: Kenar, tarama ağacında kenar tarandığında kapalı döngü oluşturuyorsa ADIM 1 yeniden uygulanır, oluşturmuyorsa ADIM 3 uygulanır.
ADIM3: Kenar taranır, kontrol edilecek kenar varsa ADIM 1 uygulanır, yoksa ADIM 4 uygulanır.
ADIM4: Tüm kenarlar kontrol edildiğinden asgari tarama ağacı oluşturma işlemi tamamlanır.

Zaman Karmaşıklığı: O(ElogV)

E kenar sayısı, V düğüm sayısı olmak üzere, kenarların sıralanması işlemi O(ElogE) zaman alacaktır. Bulma ve birleştirme işlemi için ise O(logV) zaman gereklidir. Toplam zaman O(ElogE) + O(logV) olarak alındığında, E ve V aynı kabul edilerek kabaca karmaşıklık **O(ElogV)** olarak ifade edilebilir.

2.c. Binary Search Tree (İkili Arama Ağacı)

Pratik uygulamalarda hızlı sonuç gerektiren aramalar için verilerin ikili arama ağacı ile tutularak hızlıca ulaşılması sağlanabilir. Aşağıda ele alınan örnekte verilerin sıkıştırılarak tutulma yöntemi bulunmaktadır.

Günlük Yaşam Kullanımı Örneği: "Veri Sıkıştırma"

JPEG ve MP3 gibi verilerin sıkıştırılmasında huffman sıkıştırma algoritması kullanılabilmektedir. Bir düz metin için huffman algoritması şöyle olmaktadır:

1.ADIM: Her karakter için bir düğüm oluşturulur. Düğümde karakter ile birlikte, karakterin tekrar etme sayısı, sağ ve sol düğümler tutulur.
2.ADIM: Bu düğümler karakterlerin tekrar etme sayılarına göre minheap'e atılır.
3.ADIM: Yığındaki en az tekrar eden iki karakterin düğümü listeden çıkarılır.
4.ADIM: Çıkarılan düğümlerin karakter tekrar etme sayıları toplamlarından bir düğüm oluşturulur ve bu düğüme çıkarılan iki düğümden frekansı düşük olan sol çocuk büyük olan sağ çocuk olacak şekilde düğümler eklenir.
5.ADIM: Yeni düğüm daha sonra sıralı listeye eklenir. Listede tek düğüm kalana kadar 3-5 adımları devam eder. Listede kalan tek düğüm Ağacının kök düğümü olmaktadır.

Zaman Karmaşıklığı: O(nlogn)

Her işlemde yeni düğümü eklemek O(logn) süresi gerektirir. Bu işlem n eleman için n kez tekrarlanmakta ve dolayısıyla zaman karmaşıklığı **O(nlogn)** olmaktadır.