Ramazan Göl 221110032 - Veri Yapıları Bütünleme Ödevi

Selection Sort  
  
Verinin hafızada sıralı tutulması için geliştirilen sıralama algoritmalarından (sorting algorithms) bir tanesidir. Basitçe her adımda dizideki en küçük sayının nerede olduğu bulunur. Bu sayı ile dizinin başındaki sayı yer değiştirilerek en küçük sayılar seçilerek başa atılmış olur.

Sıralanmak istenen verimiz:  
  
5,7,2,9,6,1,3,7  
  
olsun. Bu verilerin bir oluşumun belirleyici alanları olduğunu düşünebiliriz. Yani örneğin vatandaşlık numarası veya öğrenci numarası gibi. Dolayısıyla örneğin öğrencilerin numaralarına göre sıralanması durumunda kullanılabilir.

Seçerek sıralamanın çalışması yukarıdaki bu örnek dizi üzerinde adım adım gösterilmiştir.

0. adım: başlangıç adımı i=0 olarak ata.

1. adım: dizideki i. sırasından sonraki en küçük sayının yerini bul. Bu dizideki en küçük sayı 1’dir ve 5. sıradadır.

2. adım dizinin i. sırasındaki sayıyı bu en küçük sayı ile yer değiştir: 1,7,2,9,6,5,3,7

3. adım : i’yi bir arttır ve 1. adıma git.

Dolayısıyla 3. adımdan sonra i=1 olacak ve sonra ilk sayı olan 1 atlanaraka kalan sayılar olan 7,2,9,6,5,3,7 sayıları arasından en küçük sayı bulunur. Bu sayı 2’dir ve sırası da 2dir. sıradaki sayı olan yerdeki sayı ile yer değiştirir ve sonuç : 1,2,7,9,6,5,3,7 olarak bulunur.

Artık i değeri 2dir ve bu sayıdan itibaren en küçük sayı 7,9,6,5,3,7 arasında aranır. bulunan 3’ün sırası 6’dır. Bu sayı da sıradaki yerini alır ve sonuç : 1,2,3,9,6,5,7,7 olur. Bu işlem böylece devam eder ve dizinin değişimi aşağıda gösterilmiştir:

1,2,3,5,9,6,7,7

1,2,3,5,6,9,7,7

1,2,3,5,6,7,9,7

1,2,3,5,6,7,7,9

olarak bulunur.

Merge Sort

Verinin hafıza da sıralı tutulması için geliştirilen sıralama algoritmalarından bir tanesidir. Basitçe sıralanacak olan diziyi ikişer elemanı kalan parçalara inene kadar sürekli olarak ikiye böler. Sonra bu parçaları kendi içlerinde sıralayarak birleştirir. Sonuçta elde edilen dizi sıralı dizinin kendisidir.

Sıralanmak istenen verimiz:

5,7,2,9,6,1,3,7

olsun. Bu verilerin bir oluşumun belirleyici alanları olduğunu düşünebiliriz. Yani örneğin vatandaşlık numarası veya öğrenci numarası gibi. Dolayısıyla örneğin öğrencilerin numaralarına göre sıralanması durumunda kullanılabilir.

Birleştirme sıralamasının çalışması yukarıdaki bu örnek dizi üzerinde adım adım gösterilmiştir. Öncelikle parçalama adımları gelir. Bu adımlar aşağıdadır.

1. adım diziyi ikiye böl:

5,7,2,9 ve 6,1,3,7

2. adım çıkan bu dizileri de ikiye böl:

5,7 ; 2,9 ; 6,1 ; 3,7

3. adım elde edilen parçalar 2 veya daha küçük eleman sayısına ulaştığı için dur (aksi durumda bölme işlemi devam edecekti)

4. adım her parçayı kendi içinde sırala

5,7 ; 2,9 ; 1,6 ; 3,7

5. Her bölünmüş parçayı birleştir ve birleştirirken sıraya dikkat ederek birleştir (1. ve 2. parçalar ile 3. ve 4. parçalar aynı gruptan bölünmüştü)

2,5,7,9 ve 1,3,6,7  
6. adım, tek bir bütün parça olmadığı için birleştirmeye devam et

1,2,3,5,6,7,7,9

7. adım sonuçta bir bütün birleşmiş parça olduğu için dur. İşte bu sonuç dizisi ilk dizinin sıralanmış halidir.

Quick Sort

Verinin hafızada sıralı tutulması için geliştirilen sıralama algoritmalarından bir tanesidir. Basitçe sıralanacak olan dizideki orta noktada (mean) bulunan bir sayıyı seçerek diğer bütün sayıları bu orta sayıdan büyük veya küçük diye sınıflayarak sıralama yapmayı hedeflemektedir. Ayrıca bu seçilen orta noktaya eksen (pivot) adı da verilir çünkü bütün diğer sayılar bu sayının ekseninde sıralanacaktır.

Sıralanmak istenen verimiz:

5,7,2,9,6,1,4,7

olsun. Bu verilerin bir oluşumun(composition) belirleyici alanları olduğunu düşünebiliriz. Yani örneğin vatandaşlık numarası veya öğrenci numarası gibi. Dolayısıyla örneğin öğrencilerin numaralarına göre sıralanması durumunda kullanılabilir.

Yukarıda verilen bu örnek dizinin sıralanması adım adım aşağıda anlatılmıştır:

1. adım, dizinin ortasındaki sayı bulunur. Bu örnekte 8 sayı olduğu için ortadaki sayı 4. elemandır. Bu elemanın değeri de 9’dur. Bu durum aslında biraz bahtsız bir durumdur çünkü tesadüfen dizideki en büyük sayıdır. Bu mesele 2. adımda ortaya çıkacaktır.

2.adım: Diziden 1.adımda seçilen sayıya göre dizideki bütün elemanları küçük veya büyük diye sınıflandır. Tabi 1. adımda şanssız bir şekilde en büyük sayı seçildiği için bütün sayılar 9’dan küçük olarak sınıflandırılacaktır.

5,7,2,6,1,4,7 (9)

3. adım: Sınıflandırılana küçük ve büyük dizileri tekrar hızlı sıralamaya ver. Yani 5,7,2,6,1,4,7 dizisini aynı adımlarla tekrar sıralıyoruz.

4. adım: eleman sayımız 7 ve ortadaki eleman 3. eleman olan 6 olur. Dizideki sayılar 6’dan büyük ve 6’dan küçük diye sınıflandırılırsa:

5,2,1,4 (6) 7,7

olarak iki grup elde edilir. Bu grupları da sıralamak üzere tekrar hızlı sıralama algoritmasına veririz. Dolayısıyla 5,2,1,4 sayıları ayrı ve 7,7 sayıları ayrı ayrı sıralanacaktır.

5. adım 5,2,1,4 sayılarının orta değeri 2’dir ve sınıflandırılırsa:

1 (2) 4,5 bulunur. Aynı zamanda diğer dizi olan 7,7 sıralanırsa sonuç değişmez ve 7,7 bulunur.

6. adım 1 sıralanırsa 1 ve 4,5 sıralanırsa 4,5 bulunur.

7.adım. Bu adımdan sonra artık birleştirme işlemine geçilebilir. Buna göre 6. adımda sıralanan değerleri birleştirirsek

1,2,4,5 değerleri elde edilir.

8.adım: 4. adımdaki sayılar birleştirilirse 1, 2 , 4 , 5, (6),7,7 sayıları elde edilir.

9.adım: 2. adımdaki sayılar birleştirilirse 1,2,4,5,6,7,7 (9) olarak dizinin sıralanmış hali elde edilir.

Bubble Sort

Verinin hafızada sıralı tutulması için geliştirilen sıralama algoritmalarından bir tanesidir. Basitçe ardışık duran iki hafıza bloğunun birbirine nispetle sıralanması ve bu işlemin sürekli tekrarlanması sayesinde sıralar. Ardışık iki hafıza bloğuna bakmasından dolayı baloncuk ismini almıştır. Çünkü bu bakma işlemi bir baloncuğun (bubble) hareket etmesi gibi sayıların üzerinde hareket etmektedir.

Sıralanmak istenen verimiz:

5,7,2,9,6,1,3

olsun. Bu verilerin bir oluşumun(composition) belirleyici alanları olduğunu düşünebiliriz. Yani örneğin vatandaşlık numarası veya öğrenci numarası gibi. Dolayısıyla örneğin öğrencilerin numaralarına göre sıralanması durumunda kullanılabilir.

Kabarcık sıralama algoritmasının çalışması adım adım gösterilmiştir:

1.adım : 5,2,7,6,1,3,9  
  
2.adım: 2,5,6,1,3,7,9  
  
3. adım: 2,5,1,3,6,7,9  
  
4. adım: 2,1,3,5,6,7,9  
  
5. adım:1,2,3,5,6,7,9  
  
6. adım:1,2,3,5,6,7,9

Insertion Sort

Insertion Sort, programlaması oldukça basit ancak performansı merge sort,quick sort gibi sıralamalara göre nispeten yavaş bir sıralama algoritmasıdır.  
  
Sıralanacak olan sayılarımız :

33 44 21 83 56 73 22 olsun. Bu sayıları sıralamaya ilk sayıdan başlıyoruz (yani 33).

İlk geçişte sadece 33 sayısı sıralanmış oluyor :

33| 44 21 83 56 73 22 ( | işareti o anda kadar sıraladığımız sayıları gösteriyor. Bu işaretin “|” solundaki sayılar sıralanmış kabul ediyoruz. Ve her geçişte bir sağındaki sayıyı alıyoruz)

İkinci geçişte sıralayacağımız sayı 44. 33 ile 44’ü karşılaştırıyoruz 33 küçük dolayısıyla yer değiştirmiyorlar:

33 44 | 21 83 56 73 22

Üçüncü geçişte sıradaki sayı 21. 21 ile 44 karşılaştırılıyor ve 21 küçük olduğu için 44 ile yer değiştiriyorlar :

33 21 44 | 83 56 73 22 (geçişimiz henüz bitmedi çünkü 21, 33’ten de küçük:)

21 33 44 | 83 56 73 22 (şimdi 3. geçişi tamamladık ve bir sonraki sayı 83’ü alabiliriz:)

Dördüncügeçişte 83 var:

21 33 44 83 | 56 73 22 (bu geçiş çabuk bitti çünkü 83, 44’ten büyük ve sadece bunu görmemiz durmamız için yeterli sonuçta 56’ya kadar sıralamış olduk)

Beşinci geçişte 56’yı sıralayacağız:

21 33 44 56 83 | 73 22 (burada 56 ile 83 karşılaştırıldı ve 56 sayısı 83’ün soluna kaydırıldı. Bunun sebebi 56’nın 83’ten küçük olması. Bu geçişte burada bitti çünkü 56, 44’ten büyüktür)

Altıncı geçiştesıra 73’te

21 33 44 56 73 83 | 22 (sıralamamız yine tek adımda bitiyor çünkü 73, 83’ten küçük ve 56’dan büyük)

Yedinci geçişte 22 sayısını yerleştireceğiz ve adım adım 22’den küçük olan bir sayı görene kadar 22’yi dizide kaydırıyoruz:

21 33 44 56 73 22 83 |

21 33 44 56 22 73 83 |

21 33 44 22 56 73 83 |

21 33 22 44 56 73 83 |

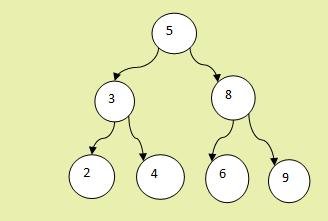
21 22 33 44 56 73 83 |

Sonuçta işaretimiz “|” dizinin sonuna ulaştı ve dizimiz sıralanmış oldu.

İkili Arama Ağaçları

İkili ağaçların (Binary Tree) özel bir hali olan ikili arama ağaçlarında, düğümlerde duran bilgilerin birbirine göre küçüklük büyüklük ilişkisi bulunmalıdır. Örneğin tam sayılardan oluşan veriler tutulacaksa bu verilerin aralarında küçük-büyük ilişkisi bulunmaktadır.

İkili arama ağacı, her düğümün solundaki koldan ulaşılabilecek bütün verilerin düğümün değerinden küçük, sağ kolundan ulaşılabilecek verilerin değerinin o düğümün değerinden büyük olmasını şart koşar.



Örneğin yukarıda bir ikili arama ağacı tasvir edilmiştir. Bu ağaçta dikkat edilecek olursa kökün solunda bulunan bütün sayılar kökten küçük ve sağında bulunan bütün sayılar kökten büyüktür.

İkili arama ağacına bu kurala uygun olarak ekleme çıkarma veya silme işlemleri yapılabilir. Ancak yapılan her işlemden sonra ikili arama ağacının yapısı bozulmamış olmalıdır.

İkili arama ağacında arama işlemi:

Yukarıda da anlatıldığı üzere ağacın üzerinde duran verilerin özel bir şekilde sıralı bulunması gerekir. Buna göre herhangi bir değeri arayan kişi ağacın kök değerine bakarak aşağıdaki 3 eylemden birisini yapar:

1. Aranan sayı kökteki değere eşittir -> Aranan değer bulunmuştur

2. Aranan sayı kökteki değerden küçüktür -> Kökün sol kolu takip edilir

3. Aranan sayı kökteki değerden büyüktür -> Kökün sağ kolu takip edilir

AVL Ağaçları

AVL Ağaçları sürekli olarak dengeli olan ikili arama ağaçlarındandır G.M. Adelson-Velsky ve E.M. Landis tarafından geliştirilmiş olan bu ağaç algoritmasının ismi de bu kişilerin isimlerinin baş harflerinden oluşmaktadır.

Algoritma basitçe, bir düğümün kolları arasındaki derinlik farkı 2 ise bu durumda dengeleme işlemi yapılır. Şayet fark 2’den az ise (yani 1 veya 0) ise bu durumda bir dengeleme işlemine gerek yoktur.

Algoritmanın ağacı dolaşma (traverse) algoritması ikili arama ağaçları ile aynıdır. Ancak ağaca ekleme ve silme işlemleri sırasında ağacın dengesinin bozulması söz konusu olduğu için bu fonksiyonlara ilave olarak derinlik kontrolü eklenmiştir.

Ekleme ve silme işlemlerinde ikili arama ağacının ekleme ve silme işleminin aynısını yaptıktan sonra aşağıdaki dengeleme işlemi yapılır.

Ağaçta ekleme veya silme yapılan her düğüm için:  
1.Adım : Sol <- Düğümün sol kolunun derinliğini ölç  
2.Adım : Sağ <- Düğümün sağ kolunun derinliğini ölç  
3.Adım : Şayet sol - sağ >1   
4.Adım : Sola dengele  
5..Adım : Şayet sağ - sol < -1   
6.Adım : Sağa dengele

B-Ağaçları  
  
İsminin nereden geldiği (B harfinin) tartışmalı olduğu bu ağaç yapısındaki amaç arama zamanını kısaltmaktır. Buna göre ağacın her düğümünde belirli sayıda anahtar veya kayıt tutularak arama işleminin hızlandırılması öngörülmüştür.

Arama hızının artmasına karşılık silme ve ekleme işlemlerinin nispeten yavaşlaması söz konusudur.

B-Ağacının tanımı

Bir B-Ağacı aşağıdaki özelliklere sahip olmalıdır:

Her [düğümün (node)](http://www.bilgisayarkavramlari.com/2008/08/01/dugum-node/) en fazla m çocuğu bulunmalıdır. (Bu sayının üzerinde eleman bulunursa düğümün çoğaltılması gerekir)

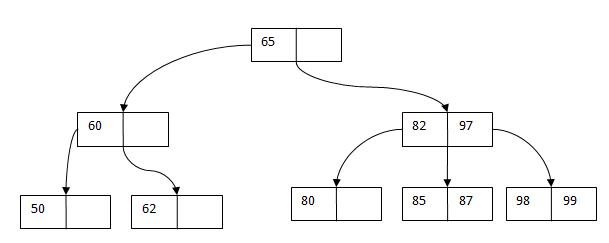
Kök ve yaprak düğümleri haricindeki her düğümün en az m/2 adet elemanı bulunmalıdır. (Bu sayının altında eleman bulunursa düğüm kaldırılır)

Bütün yapraklar aynı seviyede olmak zorundadır. Bir yaprağın seviyesinin düşmesi durumunda (daha yukarı çıkması veya daha sığ olması durumunda) ağaçta yapısal değişiklik gerekir.

Herhangi bir düğümde k çocuk bulunuyorsa k-1 elemanı gösteren anahtar (key) bulunmalıdır.

Örnek B-Ağacı

Aşağıda örnek bir b ağacı gösterilmiştir:



Aşağıda b ağacı üzerinde yapılan ekleme, silme ve arama işlemleri açıklanmıştır.

B Ağacında Arama

B-ağacında arama işlemi kökten başlar. Aranan sayı kök düğümde bulunamaması halinde arama işlemi kökte bulunan anahtarların sağında solunda veya arasında şeklinde yönlendirilir. Örneğin yukarıdaki B-ağacında 87 anahtarı aranıyor olsun. Arama işlemi için aşağıdaki adımlar gerekir:

1. Kök düğüme bakılır. 87 değeri 65’ten büyüktür. Kök düğümde tek anahtar olduğu için 65’in sağındaki [gösterici(pointer)](http://www.bilgisayarkavramlari.com/2007/10/16/pointer-gosterici-2/) takip edilir.

2. 65. sağındaki düğüme gidilir ve ilk anahtar olan 82 ile aranan anahtar olan 87 karşılaştırılır. 87 değeri 82’den büyüktür. Öyleyse ikinci anahtar olan 97 ile karşılaştırılır. 87 bu değerden küçük olduğu için bu düğümde 82 ile 97 arasında bulunan gösterici izlenir.

3. Son olarak 82 ile 97 arasındaki düğüm izlenerek ulaşılan düğümdeki anahtar ile 87 karşılaştırılır. Bu düğümdeki ilk anahtar 85’tir. 87 bu değerden büyüktür. Düğümdeki bir sonraki anahtar alınır ve 87 değeri bulunur.

B-ağaçlarının bir özelliği ağacın her düğümündeki anahtarların sıralı oluşudur. Bu yüzden bir düğümde istenen anahtar aranırken, düğümde bulunan sayılara [teker teker bakılır (linear search, doğrusal arama)](http://www.bilgisayarkavramlari.com/2008/11/09/dogrusal-arama-linear-search/)

B Ağacına Ekleme

B ağaçlarında veri yaprak düğümlerden gösterilir (pointer). Yani aslında veri ağacın düğümlerinde değil son düğümlerden gösterilen hafıza bölmeleri (RAM veya Dosya) olarak tutulur. Dolayısıyla B ağacında ekleme işlemi sırasında anahtarlar üzerinden arama ve değişiklikler yapılır ve nihayetinde amaç B ağacının son düğümleri olan yaprak düğümlerden veriyi göstermektir.

B ağaçlarında veri eklemek için aşağıdaki adımlar takip edilir:

1. Ağaçta eklenecek doğru yaprak düğümü aranır. (Bu işlem için bir önceki adımda anlatılan arama algoritması kullanılır)

2. Şayet bulunan yaprak düğümde azami anahtar sayısından (maximum key number) daha az eleman varsa (yani anahtar eklemek için boş yer varsa) bu düğüme ekleme işlemi yapılır.

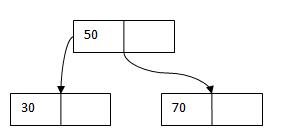
3. Şayet yeterli yer yoksa bu durumda bulunan bu yapra düğüm iki düğüme bölünür ve aşağıdaki adımlar izlenir:

1. Yeni eleman eklendikten sonra düğümde bulunan anahtarlar sıralanır ve ortadaki elemandan bölünür. (median değeri bulunur)

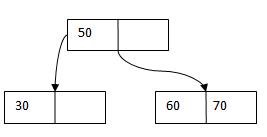
2. Ortanca değerden büyük elemanlar yeni oluşturulan sağ düğüme ve küçük elemanlar da sol düğüme konulur.

3. Ortanca eleman (median) ise bir üst düğüme konulur.

Yukarıdaki ekleme işlemini aşağıdaki örnek ağaç üzerinden görelim.

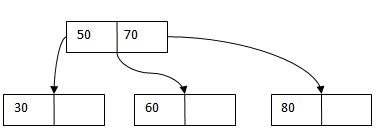


Örneğin azami anahtar sayısıs 2 olan yukarıdaki örnek ağaçta ekleme işlemi yapalım ve değer olarak 60 ekleyelim:



Yukarıdaki ekleme işlemi, ekleme algoritmamızdaki 2. durumda gerçekleşmektedir. Yani anahtarımızın ekleneceği yaprakta boş yer bulunmaktadır ve buraya yeni anahtarı ekleriz.

Şayet yukarıdaki ağaca 80 değerini ekleyecek olsaydık bu durumda da algoritmamızdaki 3. ihtimal gerçekleşmiş olacaktı.



Görüldüğü üzere 80 anahtarının ekleneceği düğüm dolmuş ve azami 2 anahtar olamsı gerekirken 3 anahtar olmuştur. Ortanca değer (median) 70 olan bu ekleme işleminden sonra ortanca değer bir üst düğüme çıkmış ve iki farklı düğüme ortanca elemanın solundaki ve sağındaki değerler yukarıdaki şekilde dağıtılmıştır.

B-Ağacından Silme

B ağacı yukarıdaki özellikleri bölümünde anlatılan özelliklerin bozulmaması için silme işlemi sırasında aşağıdaki iki yöntemden birisini izler:

1. çözümde ağaçtan ilgili anahtar bulunup silinir ve bütün ağaç yeniden inşa edilir

2. çözümde ağaçtan ilgili anahtar bulunup silinir ve bulma işlemi sırasında geçilen ağacın kısımları yeniden inşa edilir.

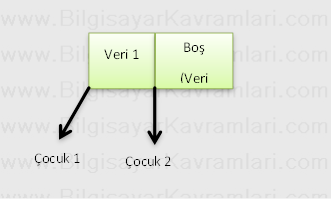
Ayrıca B+ ağacı ve B# ağacı şeklinde alt çeşitleri de bulunmaktadır.

2-3-4 Ağaçları

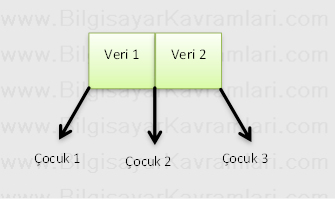
2-3-4 ağacı, B-ağaçlarının (B-Trees) özel bir halidir. Bu ağacın özelliği, düğüm boyutunun (node size) 3 ile sınırlı olmasıdır. Ağaç ayrıca sürekli olarak dengeli bir ağaç garantisi verir (balanced tree). 2-3-4 ağaçları, kırmızı siyah ağaçlarının (red-black trees) , eş şekillisi (isomorphic) olarak da düşünülebilir.

2-3-4 ağacının ismi, ağaçtaki düğümlerin değişik durumlarda değişik numaralar almasından kaynaklanmaktadır. Yani bir düğüm 2,3 veya 4 durumunda olabilir. Bu durumlara göre farklı uygulamalar söz konusudur.

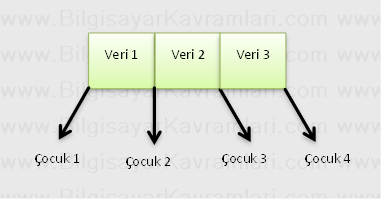
2 düğümü: 2 adet çocuk düğümü gösteren gösterici (pointer) ve bir veriden oluşur



3 düğümü: 3 adet çocuk düğümü gösteren gösterici (pointer) ve iki veriden oluşur



4 düğümü: 4 adet çocuk düğümü gösteren gösterici (pointer) ve üç veriden oluşur



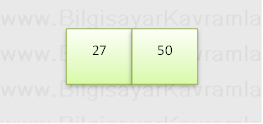
Ekleme işlemini bir örnek üzerinden anlamaya çalışalım

50, 27, 97, 52, 19, 11, 111

Bu sırayla verilen değerleri sırasıyla ağaca ekleyelim:

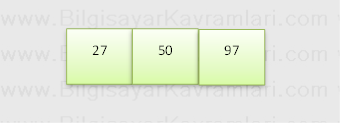


İlk sayı olan 50’nin eklenmesi sonucunda ağaçta tek bir düğüm ve bu düğümde tek bir veri bulunuyor. Bu tip düğüme 2 düğümü ismi verilmektedir ve iki çocuğu da boştur (null).

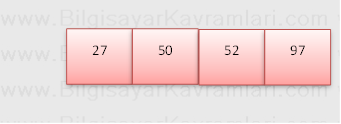


İkinci eklenen değer 27’dir. Ağacımızda tek düğüm bulunduğundan ve bu düğüm 2 tipinde olduğundan, düğümde yeni değeri alacak yer bulunmaktadır. Bu yere yeni gelen değer yerleştirilir. Elbette düğümlerin içerisinde sıra bulunması gerektiği için, 27 değeri, 50 değerinin soluna yerleştirilmiştir. Bu kural bundan sonraki ekleme işlemlerinde de geçerli olacaktır. Yani düğüm içerisindeki değerler kendi aralarında sıralı olacak ve herhangi bir değerin solundaki çocuklarının değeri, kendi değerinden küçük ve sağındaki değeri kendi değerinden büyük olacaktır.

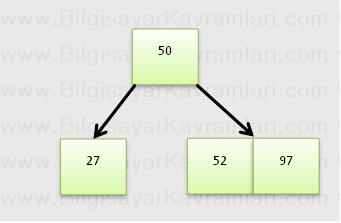
Ardından gelen ekleme değeri, 97’dir. Bu değer düğüme eklenince, aşağıdaki gibi bir yapı elde edilir:



Yukarıdaki bu yeni yapı, bizim 2-3-4 ağacımızdaki 4 tipindeki yapıdır ve kabul edilebilir bir durumdur. Ancak bir sayı daha eklenince yapı bozulacaktır. Nitekim sıradaki sayımız olan 52’nin eklendiğini düşünelim:

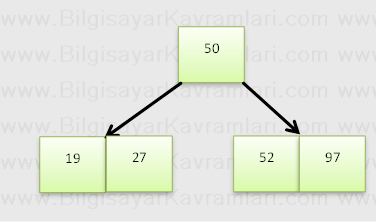


Şimdiye kadar öğrendiklerimize göre, yukarıdaki gibi bir yapı olması beklenir. Ancak yukarıdaki yapı 2-3-4 ağacı tarafından kabul edilemez (2,3 veya 4 tipinde bir düğüm değildir). Dolayısıyla bu yapının kabul edilir bir şekle indirgenmesi gerekir.



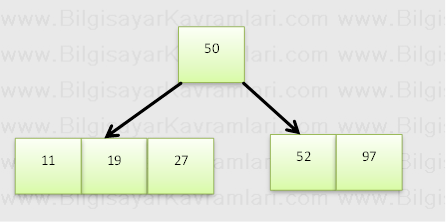
Yukarıdaki yeni şekilde, ağaç 50 ve 27 değerlerini taşıyan iki adet 2 tipinde düğüm ve bir adet 52 ve 97 değerlerini taşıyan 3 tipinde düğüme indirgenmiştir. Bu düğümlerin tamamı 2-3-4 ağacı tarafından kabul edilir düğümlerdir.

Ekleme işlemine 19 sayısı ile devam edelim.

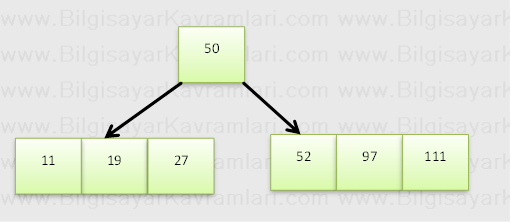


Yukarıdaki şekilde görüldüğü üzere, 19 sayısı, 50’den küçük olduğu için kökteki bu değerin solundaki düğüme yerleştirilmiştir.

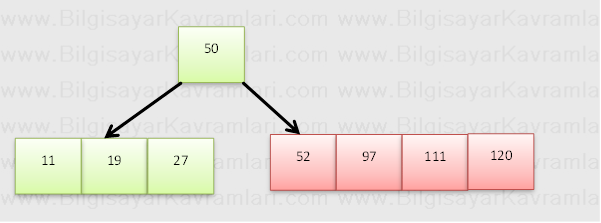
11 sayısı eklendikten sonra, bu sayı da daha önceden olduğu gibi, 50’nin solundaki düğüme sıralı olarak eklenir.



Benzer şekilde 111 sayısının eklenmesi de kök değerin sağına yapılır:

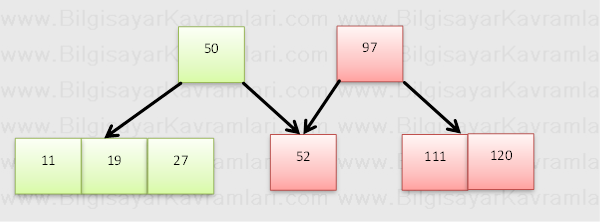


Son olarak ağacın 3 düğümü olarak taşıyamayacağı bir örneği göstermek için, örneğin 120 sayısını ağaca eklemeye çalışalım:

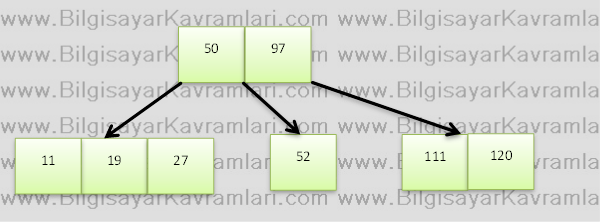


Normalde 120 sayısının ekleneceği yer, yukarıdaki şekilde gösterildiği gibi ağacın sağ koludur. Ancak bu durum 2-3-4 ağacı tarafından kabul edilemez çünkü herhangi bir düğüm yapısına uygun değildir.

Çözüm olarak daha önceden 52 sayısının eklendiği örnekte olduğu gibi düğümleri indirgiyoruz.



Bu işlemden sonra kök düğümünü güncelleyerek sonuçta çıkacak olan ağacı elde ediyoruz

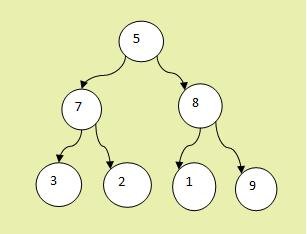


Ağacımız son halinde 2-3-4 ağacı kurallarına uygun düğümlerden oluşmaktadır.

BFS (Breath First Search)

Bir ağaç dolaşma algoritmasının (tree traverse algorithm, tree traversal) ilk önce aynı seviyede bulunan komşularını araması durumudur.

Örneğin aşağıdaki ağacı ele alalım:



Ağacı dolaşma sırlaması örneğin 5,7,8,3,2,1,9 şeklindeyse bu dolaşmaya sığ öncelikli arama ismi verilebilir.

Bu arama sıralamasında, dolaşma sıralaması aşağıdaki ihtimallerden birisi olabilir:

NLR : Node Left Right (Düğüm Sol Sağ)

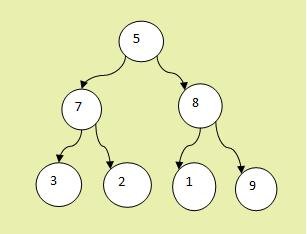
NRL : Node Right Left (Düğüm Sağ Sol)

Yani öncelikle düğüm sonra altındaki üyelere hareket edilir.

DFS (Depth-First Search)

Bir ağaç dolaşma algoritmasının (tree traverse algorithm, tree traversal) ilk önce alt seviyesinde bulunan komşularını araması durumudur.

Örneğin aşağıdaki ağacı ele alalım:



Ağacı dolaşma sırlaması örneğin 3, 2, 7, 1, 9 , 8 , 5 şeklindeyse bu dolaşmaya deep-first search ismi verilebilir.

Bu arama sıralamasında, dolaşma sıralaması aşağıdaki ihtimallerden birisi olabilir:

LRN : Left Right Node (Sol Sağ Düğüm)

RLN : Right Left Node (Sağ Sol Düğüm)

RNL : Right Node Left (Sağ Düğüm Sol)

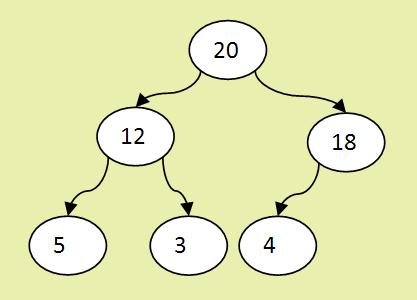
RLN : Right Left Node (Sağ Sol Düğüm)

Yani öncelikle düğüm sonra altındaki üyelere hareket edilir.

Min-Max Heap Tree

Yığın ağacı bilgisayar bilimlerinde özellikle sıralama amacıyla çokca kullanılan bir veri yapısıdır. Bu veri yapısı üst düğümün (atasının) alt düğümlerden (çocuklarından) her zaman büyük olduğu bir [ikili ağaç (binary tree)](http://bilgisayarkavramlari.com/2008/05/07/ikili-agac-binary-tree/) şeklinde düşünülebilir.

Aşağıda örnek bir yığın ağacı verilmiştir:



Yukarıdaki ikili ağaçta dikkat edilirse ağaç dengeli bir şekilde sırayla doldurulmuştur. Buna göre yeni bir eleman daha eklenmesi durumunda ağacın eleman sayısı 7’ye yükselecek ve sayıların yeri değişmekle birlikte yeni eleman şu andaki 18 sayısını içeren düğümün sağına gelecektir.

Yığın ağaçları her zaman sırayla dolar ve sırayla boşalır.

[Yukarıdaki ağacın bir dizi ile tutulması da mümkündür.](http://bilgisayarkavramlari.com/2008/08/09/dizi-uzerinde-agac-kodlamasi/)

Ayrıca herhangi bir ağacı yığın ağacına çevirmek (yığınlamak, heapify) de mümkündür ve bu işlem bir yığın ağacı oluşturma işleminin temelini oluşturur. Basitçe yığın ağacına yapılan her ekleme ve her çıkarma işleminden sonra bu yığınlama (heapify) işlemi yapılarak yığın ağacının formunu koruması sağlanmalıdır.

Graflar ve Gösterimleri

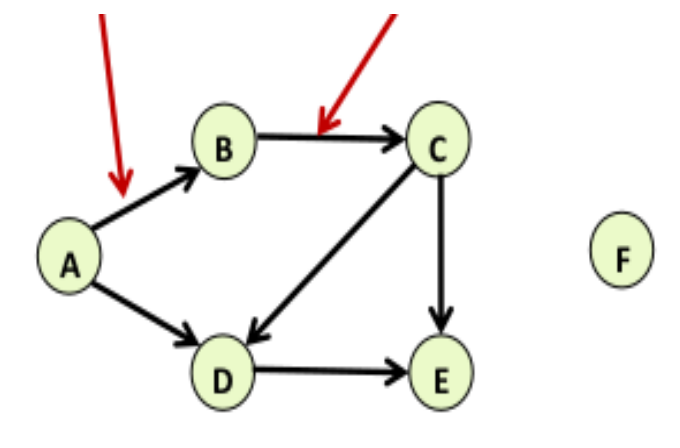
Graf, matematiksel anlamda, düğümlerden ve bu düğümler arasındaki ilişkiyi gösteren kenarlardan oluşan bir kümedir. Mantıksal ilişki, düğüm ile düğüm veya düğüm ile kenar arasında kurulur.

* Bağlantılı listeler ve ağaçlar grafların özel örneklerindendir.
* Fizik, Kimya gibi temel bilimlerde ve mühendislik uygulamalarında ve tıp biliminde pek çok problemin çözümü ve modellenmesi graflara dayandırılarak yapılmaktadır.

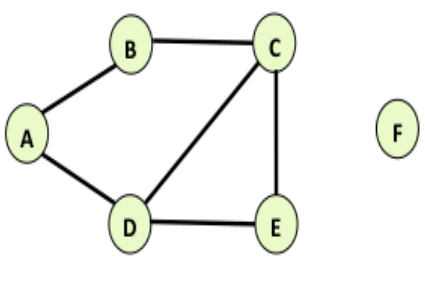
Graflar - Uygulama alanları   
1.Elektronik devreler - Baskı devre kartları (PCB), Entegre devreler   
2.Ulaşım ağları - Otoyol ağı, Havayolu ağı   
3.Bilgisayar ağları - Lokal alan ağları ,İnternet   
4.Veritabanları - Varlık-ilişki (Entity-relationship) diyagramı

Bir G grafı D ile gösterilen düğümlerden (node veya vertex) ve K ile gösterilen kenarlardan (Edge) oluşur.   
 − Her kenar, iki düğümü birleştirilerek iki bilgi (Düğüm) arasındaki ilişkiyi gösterir ve (u, v) şeklinde ifade edilir. (u, v) iki düğümü göstermektedir.   
Bir graf üzerinde n tane düğüm ve m tane kenar varsa, matematiksel gösterilimi, düğümler ve kenarlar kümesinden elamanların ilişkilendirilmesiyle yapılır:   
 D = {d0 , d1 , d2 ... dn-1 , dn } Düğümler kümesi   
 K = {k0 , k1 , k2 ... km-1 , km } Kenarlar kümesi   
 G = (D, K) → Graf

G = (D, K) grafı aşağıda verilmiştir.   
 D = {A, B, C, D, E, F}   
 K = {(A, B), (A, D), (B, C), (C, D), (C, E), (D, E)}



Komşu(Adjacent): Bir G grafı üzerindeki di ve dj adlı iki düğüm, kenarlar kümesinde bulunan bir kenarla ilişkilendiriliyorsa bu iki düğüm birbirine komşu (adjacent, neighbor) düğümlerdir; k = {di ,dj } şeklinde gösterilir ve “k kenarı hem di hem de dj düğümleriyle bitişiktir” (incident) denilir.  
  
Diğer bir deyişle k kenarı di ve dj düğümlerini birbirine bağlar veya “di ve dj düğümleri k kenarının uç noktalarıdır” denilir.   
  
− (A, B) komşudur.   
− (B, D), (C, F) komşu değildir.



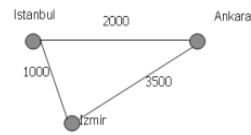
Komşuluk ve Bitişiklik: Bir G grafı komşuluk ilişkisiyle gösteriliyorsa Gdd = {(di ,dj )...}, bitişiklik ilişkisiyle gösteriliyorsa Gdk = {(di ,kj )...} şeklinde yazılır. (Gdd:Gdüğümdüğüm,Gdk:Gdüğümkenar)   
− Örneğin, şekil a) 2-düğümlü basit graf Gdd = {(d0 ,d1 )} veya Gdk = {(d0 ,k0 ),(d1 ,k0 )} şeklinde yazılabilir;   
− b) verilen 4-düğümlü basit grafta Gdd= {(d0 ,d0 ),(d0 ,d1 ),(d0 ,d2 ),(d0 ,d3 ),(d1 ,d2 ),(d2 ,d3 )} veya Gdk={(d0 ,k0 ),(d0 ,k1 ),(d0 ,k2 ),(d0 ,k3 ),(d1 ,k1 ),(d1 ,k4 ),(d2 ,k2 ),(d2 ,k4 ),(d2 ,k5 ),(d3 ,k3 ),(d3 ,k5 )} şeklinde yazılır.



Yönlendirilmiş Graf (Directed Graphs): Bir G grafı üzerindeki kenarlar bağlantının nereden başlayıp nerede sonlandığını belirten yön bilgisine sahip ise yönlü-graf veya yönlendirilmiş graf (directed graf) olarak adlandırılır.   
 − Yönlü graflar, matematiksel olarak gösterilirken her bir ilişki oval parantezle değil de <> karakter çiftiyle gösterilir.   
  
Yönlendirilmemiş Graf (Undirected Graphs)   
 − Hiçbir kenarı yönlendirilmemiş graftır.

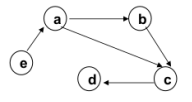
Yönlendirilmiş Kenar (Directed Edge)   
− Sıralı kenar çiftleri ile ifade edilir.   
 (u, v) ile (v, u) aynı değildir.   
− İlk kenar orijin ikinci kenar ise hedef olarak adlandırılır.   
  
Yönlendirilmemiş Kenar (Undirected Edge)   
− Sırasız kenar çiftleri ile ifade edilir.   
(u, v) ile (v, u) aynı şeyi ifade ederler

Ağırlıklı Graf (Weighted Graphs) : Graf kenarları üzerinde ağırlıkları olabilir. Eğer kenarlar üzerinde ağırlıklar varsa bu tür graflara ağırlıklı/maliyetli graf denir. Eğer tüm kenarların maliyeti 1 veya birbirine eşitse maliyetli graf olarak adlandırılmaz; yön bilgisi de yoksa basit graf olarak adlandırılır.  
  
Ağırlık uygulamadan uygulamaya değişir:   
 Şehirler arasındaki uzaklık.   
Routerler arası bant genişliği.  
 Router: Yönlendirici

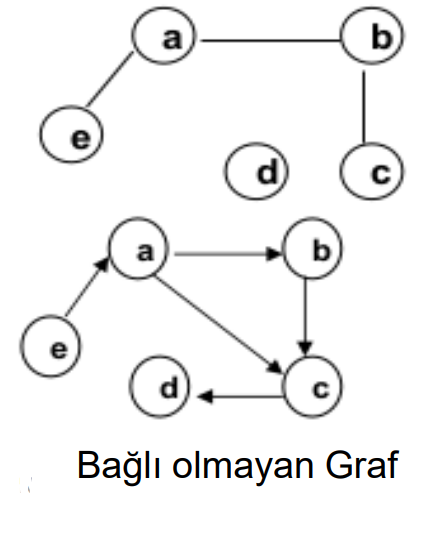
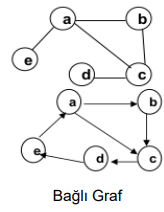


Yol (path) : Bir çizgi kümesinde yol, düğümlerin sırasıdır. Kenar uzunluğu, yolun uzunluğunu verir.   
− V1 ,V2 ,V3 ..... VN ise yol N – 1 dir.   
− Eğer yol kenar içermiyorsa o yolun uzunluğu sıfır’dır.   
− İzmir’den Ankara’ya doğrudan veya İstanbul’dan geçerek gidilebilir   
− (İzmir- İstanbul- Ankara)

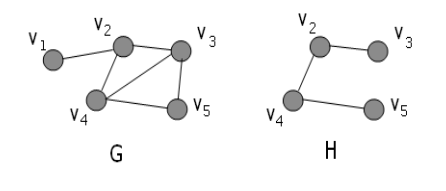
Basit Yol (Simple Path) : Bir yolda ilk ve son düğümler dışında tekrarlanan düğüm yoksa basit yol diye adlandırılır.   
− eabcd basit yol’dur fakat eabcabcd veya abcabca basit yol değildir



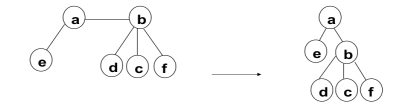
Uzunluk : Bir yol üzerindeki kenarların uzunlukları toplamı o yolun uzunluğudur.   
  
Bağlı veya Bağlı olmayan Çizge (Connected Graph):   
− Eğer bir graftaki tüm düğümler arasında en azından bir yol varsa bağlı graftır. Eğer bir grafta herhangi iki düğüm arasında yol bulunmuyorsa bağlı olmayan graftır.



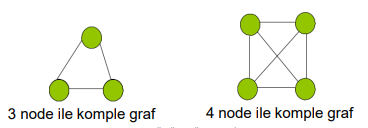
Alt Graf (Subgraph) : H çizgesinin köşe ve kenarları G çizgesinin köşe ve kenarlarının alt kümesi ise; H çizgesi G çizgesinin alt çizgesidir (subgraph).   
- G (V, E) şeklinde gösterilen bir grafın, alt grafı H(U, F) ise U alt küme V ve F alt küme E olur.



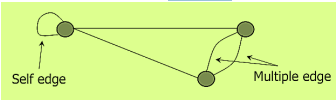
Ağaçlar(Trees) : Ağaçlar özel bir çizgi kümesidir. Eğer direk olmayan bir çizgi kümesi devirli (cycle) değilse ve de bağlıysa (connected) ağaç olarak adlandırılır.



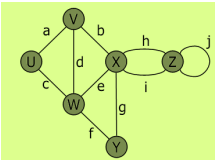
Komple Çizge(Complete Graph) : Eğer bir graftaki her iki node arasında bir kenar varsa komple graftır.



Multigraf: Multigraf iki node arasında birden fazla kenara sahip olan veya bir node’un kendi kendisini gösteren kenara sahip olan graftır.



Örnek   
a, b ve d kenarları V node’unun kenar bağlantılarıdır.   
X node’unun derecesi 5’ tir.   
h ve i çoklu (multiple) kenarlardır.   
j kendi kendisine döngüdür (self loop)

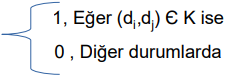


Düğüm Derecesi (Node Degree): Düğüme bağlı toplam uç sayısıdır; çevrimli kenarlar aynı düğüme hem çıkış hem de giriş yaptığı için dereceyi iki arttırır.   
  
Yönlü graflarda, düğüm derecesi giriş derecesi (input degree) ve çıkış derecesi (output degree) olarak ayrı ayrı belirtilir.

Komşuluk Matrisi (Adjacency Matrice): Düğümlerden düğümlere olan bağlantıyı gösteren bir kare matrisdir; komşuluk matrisinin elemanları ki değerlerinden oluşur. Komşuluk matrisi G dd 'nin matrisel şekilde gösterilmesinden oluşur.

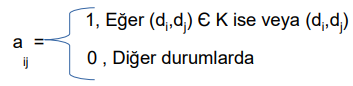
Eğer komşuluk matrisi G dd =[a ij ] ise,

Yönlü-maliyetsiz graflar için; a ij =



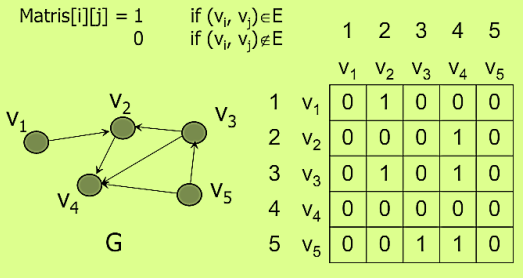
olur.

Basit (yönsüz-maliyetsiz) graflar için ise,

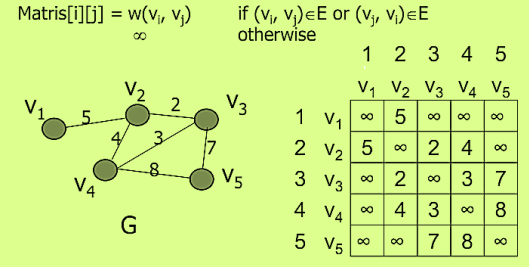


olur.

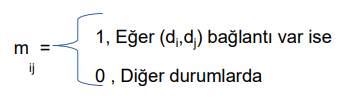
Yönlendirilmiş graf için komşu matrisi



Ağırlıklandırılmış ancak yönlendirilmemiş graf için komşu matrisi

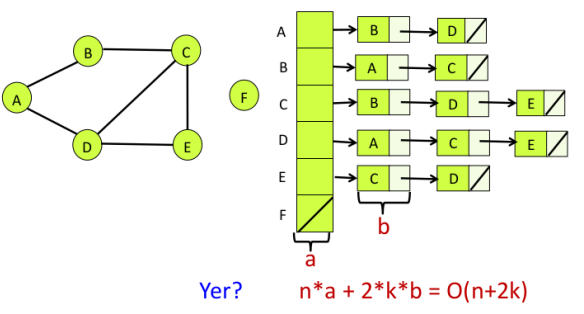


Bitişiklik Matrisi (Incedence Matrice): Düğümlerle kenarlar arasındaki bağlantı/bitişiklik ilişkisini gösteren bir matrisdir; matrisin satır sayısı düğüm, sütun sayısı kenar sayısına kadar olur. Bitişiklik matrisi G dk 'nin matrisel şekilde gösterilmesinden oluşur. Eğer bitişiklik matrisi Gdk =[mij] ise, maliyetsiz graflar için,

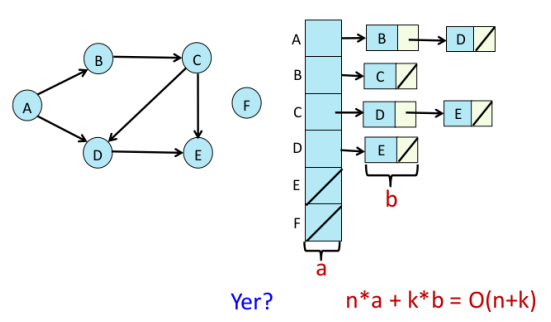


olur.

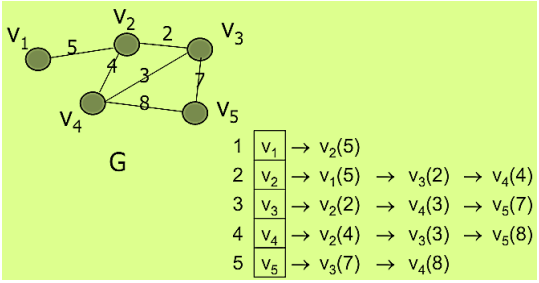
Komşuluk Listesi Gösterimi



Komşuluk Listesi (Yönlendirilmiş Graflar)



Yönlendirilmiş ve ağırlıklandırılmış graf için komşu listesi



Komşu Matrisi-Komşu Listesi

− Avantajları-dezavantajları;   
Komşu matrisi   
− Çok fazla alana ihtiyaç duyar. Daha az hafızaya ihtiyaç duyulması için sparse (seyrek matris) matris tekniklerinin kullanılması gerekir.   
− Herhangi iki node’un komşu olup olmadığına çok kısa sürede karar verilebilir.

Komşu listesi   
− Bir node’un tüm komşularına hızlı bir şekilde ulaşılır.   
− Daha az alana ihtiyaç duyar.   
− Oluşturulması matrise göre daha zor olabilir.

Not: Sparse matris; m\*n boyutlu matriste değer bulunan hücreleri x boyutlu matriste saklayarak hafızadan yer kazanmak için kullanılan yöntemin adı.