Algoritmalar

Çizge Algoritmaları

Çizge Algoritmaları

- Çizgeler karmaşıklık derecesi yüksek birçok hesaplama probleminin çözümünde kullanılmaktadır.
- Öncelikle çizge kavramları ve çizgelerin bilgisayar ortamında gösterimi ele alınacaktır.
- Arama probleminin çizge veri yapısı üzerinde nasıl çözülebileceği işlenecektir.
- Son olarak çizgelere özgü problemler ele alınacaktır.
- En küçük kapsayan ağaç ve en kısa yol algoritmaları işlenecektir.

Çizge Kavramı ve Tanımlar

- Yönlendirilmiş bir çizge (graph) G iki parametre ile tanımlanır ve $G = (E, V)$ şeklinde gösterilir.
- Bu ifadede V bir düğüm (vertex) kümesini ve $E \subseteq V \times V$ bir kenar (edge) kümesini tanımlamaktadır.
- Kenarlar düğümleri birleştirdiğinden, çizgedeki kenar sayısı için üst ve alt sınırlar aşağıdaki gibi tanımlanır:
  - $|E| = O(V^2)$.
  - Eğer G bağlı (connected) ise, $|E| \geq |V| - 1$'dir (Her düğümden diğerine yol vardır).

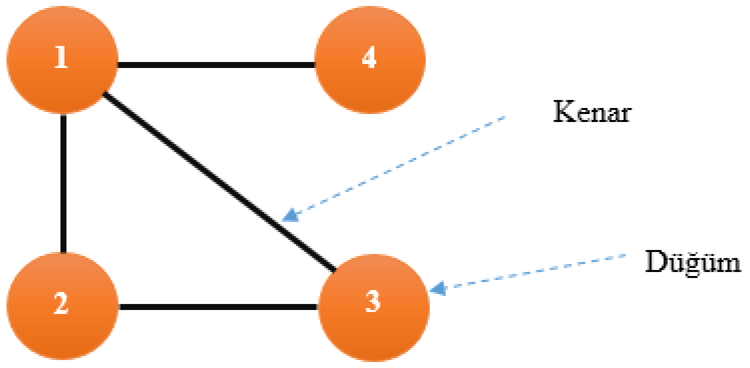
Çizge Kavramı ve Tanımlar

- Çizgelerde düğümler u ve v gibi küçük harflerle kenarlar ise (u, v) şeklinde gösterilir.
- Eğer $(u, v) \in E$ ise *u* ve *v* düğümleri komşudur. Kenarlar yönlendirilmiş ya da yönlendirilmemiş olabilir.
- Yönlendirilmiş kenarlar sıralı düğüm çiftleriyle ifade edilir. (u, v) kenarının değeri (v, u) kenarının değerine eşit değildir.
- İlk düğüm başlangıç (orijin) ve ikinci düğüm ise hedef olarak adlandırılır.
- Yönlendirilmiş çizge her kenarı yönlendirilmiş çizgedir. Yönlendirilmemiş çizgelerde ise (u, v)=(v, u)'dur.
- Yönlendirilmemiş çizge hiçbir kenarı yönlendirilmemiş çizgedir.

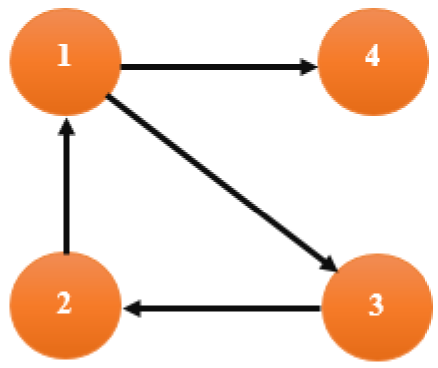
Çizge Kavramı ve Tanımlar

- Çizge içinde bir yol $v_1$den v_k ya kadar sıralı düğümleri (v_1, v_2), $(v_2$ $v_3)$, …, (v_{k-1}, v_k) kenarlarıyla birbirine bağlayan düğüm dizidir. Basit bir yolda her bir düğüm sadece bir kez bulunur. Döngü (cycle) başlangıç ve bitiş düğümleri aynı olan yola verilen isimdir. Aşağıda yönlendirilmiş ve yönlendirilmemiş örnek çizgeler verilmiştir. Örneklerde (1,2,3) düğümleri arasında bir döngü vardır.

Örnek – Yönlendirilmemiş Çizge



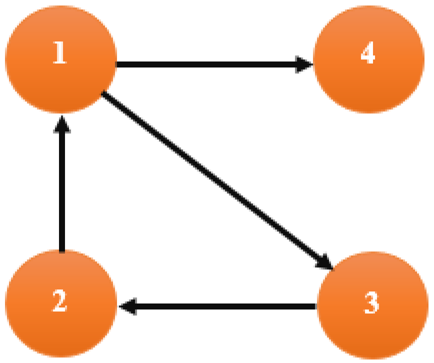
Örnek – Yönlendirilmiş Çizge



Çizge Kavramı ve Tanımlar

- Bir $G = (E, V)$ çizgesini bilgisayarda göstermenin (uygulamanın) iki standart yolu vardır: yakınlık/komşuluk matrisi ve yakınlık/komşuluk listesi.

Örnek – Yönlendirilmiş Çizge



	1	2	3	4
1	0	0	1	1
2	1	0	0	0
3	0	1	0	0
4	0	0	0	0

Komşuluk Matrisi

1	→ 3	→ 4	→ /
2	→ 1	→ /	
3	→ 2	→ /	
4	→ /		

Komşuluk Listesi

Çizge Kavramı ve Tanımlar

- Eğer bir çizgede tüm düğümler arasında en azından bir yol varsa bu bağlı bir çizgedir.
- Eğer bir çizgede herhangi iki düğüm arasında yol bulunmuyorsa bağlı olmayan (disconnected) çizgedir.
- Çizgeler birçok uygulama için kullanılabilir. Örneğin elektronik devreler, ulaşım ağları (otoyol havayolu), bilgisayar ağları vb.
- Bu uygulamalarda bir çizgede kenarların değerleri olabilir. Örneğin kenarın uzunluğuna göre yada üzerindeki yük trafiğine göre kenar değeri belirlenebilir.
- Çizge bu durumda çizge ağırlıklandırılmış çizge adını alır.
- Eğer çizgede kenar ağırlıkları belirlenmemişse her kenarın değeri 1 olarak alınır.

Çizgelerde Arama Problemi

- Graph search
- Graph traversal
- Her düğüm ziyaret edilerek aranan değer bulunmaya çalışılır
  - Ağaçlardaki tree-walk özel bir durumudur
- Genişilik Öncelikli Arama
- Derinlik Öncelikli Arama

Genişlik Öncelikli Arama (Breadth First Search)

- Genişlik Öncelikli Arama (Breadth First Search) çizgelerde kullanılabilecek en kolay arama algoritmasıdır.
- Verilen bir *s* düğümünden çizge içinde erişilebilecek tüm düğümleri bulmak için kullanılır.
- Bu algoritmada öncelikle seçilen *s* düğümünün tüm komşuları sırayla seçilir ve ziyaret edilir.
- Seçilen her komşu bir kuyruk yapısı içine atılır.
- *s* düğümünün komşusu kalmadığında kuyruk içerisindeki ilk düğüm alınır ve onun komşuları ziyaret edilerek aynı işlemler kuyruk içinde hiç bir düğüm kalmayana kadar devam eder.

Genişlik Öncelikli Arama (Breadth First Search)

BFS(*G, s*)

Adım 1. visit(*s*)

Adım 2. queue.insert(*s*)

Adım 3. **while** (queue is not empty)

Adım 4. *u* = queue.extractHead()

Adım 5. **for** each edge (*u, d*)

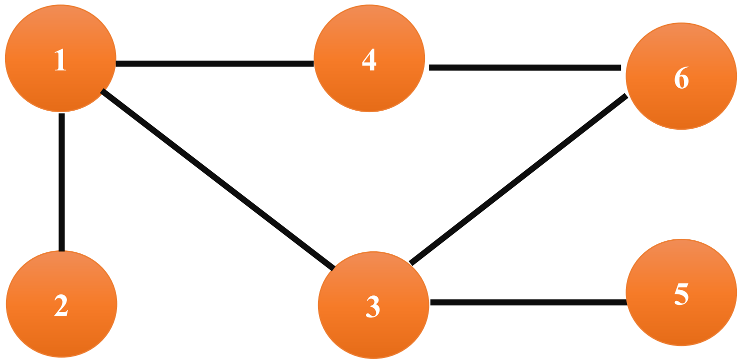
Adım 6. **if** (*d* has not been visited)

Adım 7. visit(*d*)

Adım 8. queue.insert(*d*)

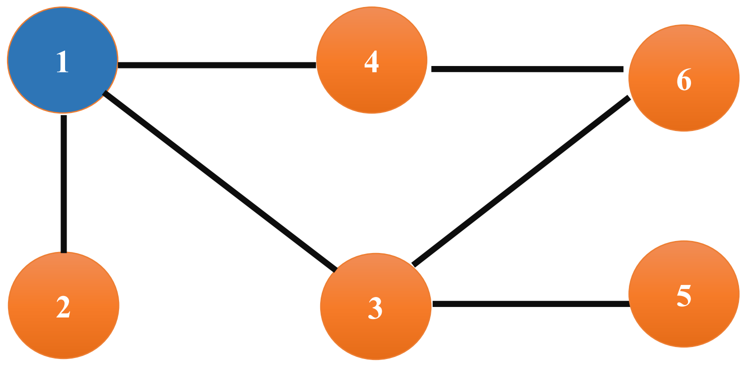
Genişlik Öncelikli Arama (Breadth First Search)

- Aşağıdaki çizgede 1 numaralı düğümden erişilebilecek tüm düğümleri bulunuz.



Genişlik Öncelikli Arama (Breadth First Search)

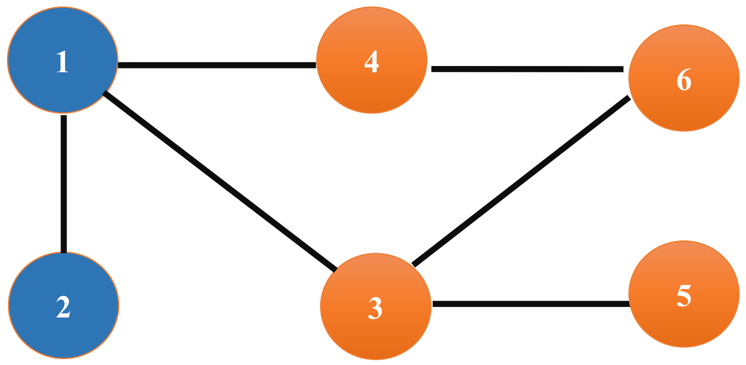
- Önce 1 numaralı düğüm ziyaret edilir. Kuyruğa alınır (Adım 2), kuyruktan çıkartılır (Adım 4) ve 1'in tüm komşuları sıra ile ziyaret edilir ve kuyruğa alınır (Adım 5-8).



- **Kuyruk: 2, 3, 4**

Genişlik Öncelikli Arama (Breadth First Search)

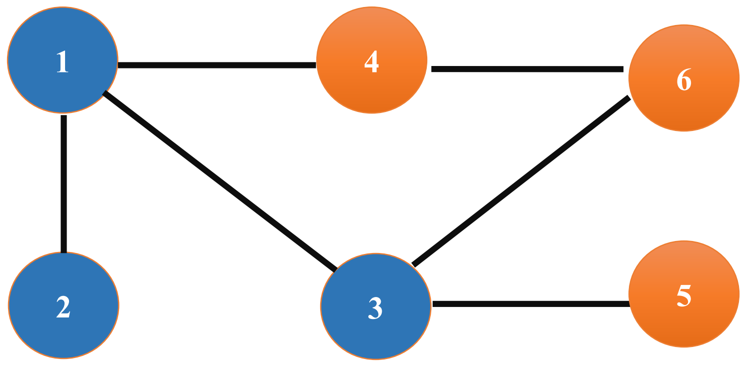
- İlk komşu olarak 2 numaralı düğüm ziyaret edilir. Kuyruktan çıkartılır (Adım 4) ve 2’in tüm komşuları sıra ile ziyaret edilir ve kuyruğa alınır (Adım 5-8).



- **Kuyruk: 3, 4**

Genişlik Öncelikli Arama (Breadth First Search)

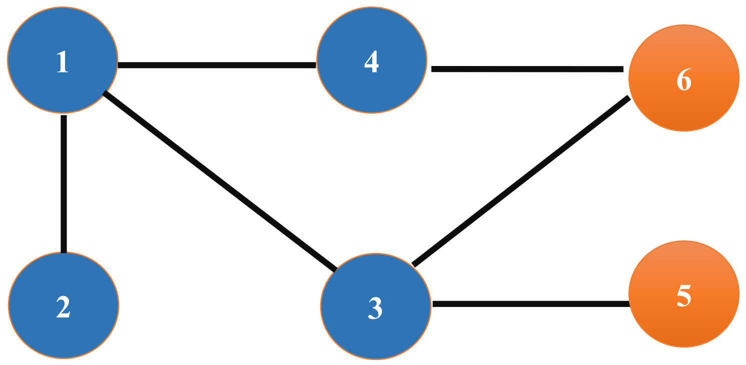
- 3 numaralı düğüm ziyaret edilir. Kuyruktan çıkartılır (Adım 4) ve 3'ün tüm komşuları sıra ile kuyruğa alınır (Adım 5-8).



- **Kuyruk: 4,5,6**

Genişlik Öncelikli Arama (Breadth First Search)

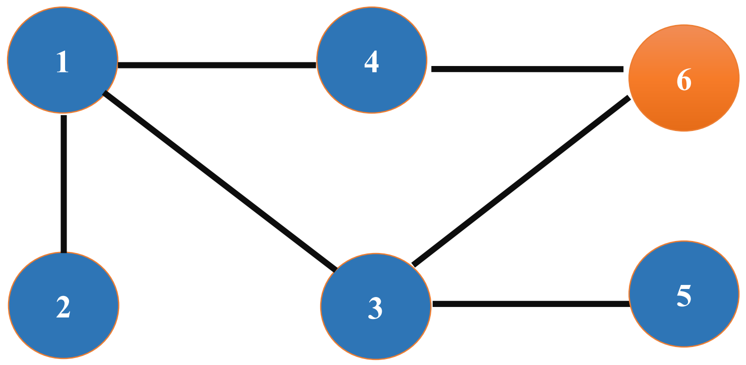
- 4 numaralı düğüm ziyaret edilir. Kuyruktan çıkartılır (Adım 4) ve 4'ün tüm komşuları sıra ile ziyaret edilir ve kuyruğa alınır (Adım 5-8). 6 numaralı düğüm daha önce kuyruğa alındığı için tekrar alınmaz.



- **Kuyruk: 5,6**

Genişlik Öncelikli Arama (Breadth First Search)

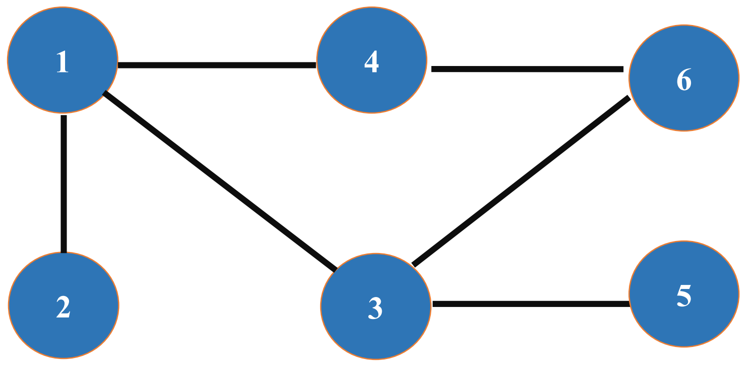
- Son iki aşamada 5 ve 6 numaralı düğümler kuyruk dışına alınır.



- **Kuyruk: 6**

Genişlik Öncelikli Arama (Breadth First Search)

- Son iki aşamada 5 ve 6 numaralı düğümler kuyruk dışına alınır.



- **Kuyruk: Boş**

Genişlik öncelikli arama algoritması çalışma zamanı analizi

- Sözde kodda görüleceği üzere algoritmada her düğüm bir kez kuyruk içine alınıp işlenmektedir.
- Bu O(V) zamanda tamamlanır. İşlenen her düğümün tüm kenarları (u, v) için işlem yapılacağı için O(E) zaman gerekir.
- Bu nedenle genişlik öncelikli arama algoritmasının toplam çalışma zamanı O(V+E) olarak belirlenir.

Derinlik Öncelikli Arama (Depth First Search)

- Verilen bir *s* düğümünden çizge içindeki ulaşılabilecek diğer tüm düğümlerin bulunması hedeflenmektedir.
- Ancak genişlik öncelikli aramadan farklı olarak tüm komşuların öncelikle bulunması yerine bir komşudan ulaşılabilecek diğer tüm düğümlerin bulunmasına öncelik verilir.
- Bir *s* düğümüne gidildikten sonra *s* düğümünün bir komşusu seçilir ve ziyaret edilir.
- Ardından onun bir komşusu seçilir ve peş peşe komşu seçimi yapılarak devam edilir.
- Komşu kalmadığında geri dönülür.

Derinlik Öncelikli Arama (Depth First Search)

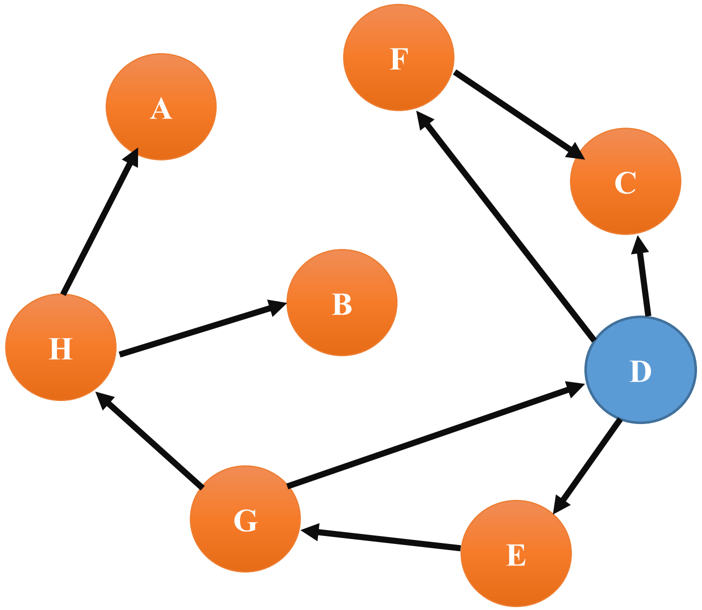
```
DOA (G,s)
for each vertex u
        visited[u] = false;
push.stack(s);
while (stack is not empty) do
        u=pop.stack();
      if (visited[u]=false) then
                visited[u] = true;
                for each unvisited neighbor v of u
                        push.stack(v)
```

Derinlik Öncelikli Arama (Depth First Search)

- Sözde koddan anlaşılacağı üzere derinlik öncelikli arama algoritmasında yığın yapısından faydalanılır.
- Öncelikle tüm düğümlerin ziyaret edilme değişkeni negatif yapılır ve başlangıç düğümü yığına eklenir.
- Yığın içinden sürekli ilk eleman alınarak ziyaret edilir. Ziyaret sırasında bu düğümün tüm komşuları da yığına eklenir.
- Kuyruk yapısının aksine yığın yapısında sürekli üstten ekleme ve çıkarma yapıldığından, derinlemesine bir ilerleme sağlanmış olur.

Derinlik Öncelikli Arama (Depth First Search)

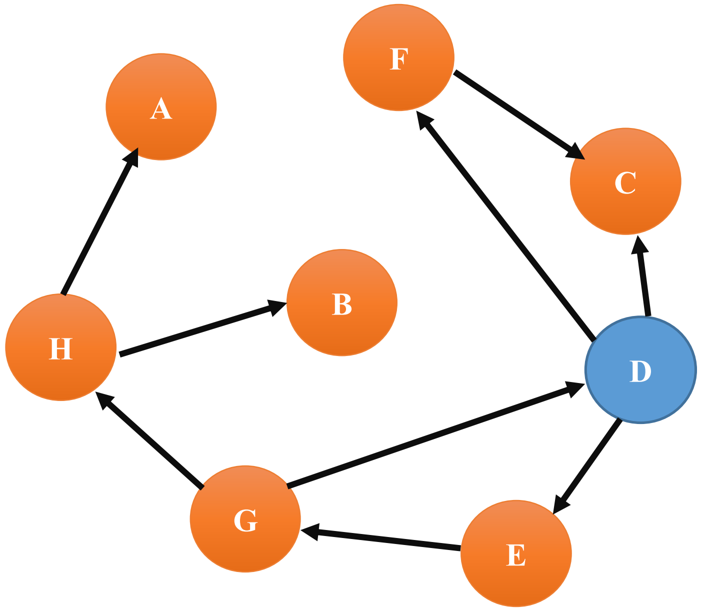
- D düğümden başlayarak derinlik öncelikli arama işlemini uygulayınız.



	Ziyaret edilen	Yığın
A		
B		
C		
D	x	
E		
F		
G		
H		D

Derinlik Öncelikli Arama (Depth First Search)

- D düğümünün komşuları yığına eklenir ve en üstteki seçilerek devam edilir (yığına ekleme sırasında alfabetik sıra takip edilecektir).



A	
B	
C	
D	x
E	
F	
G	
H	

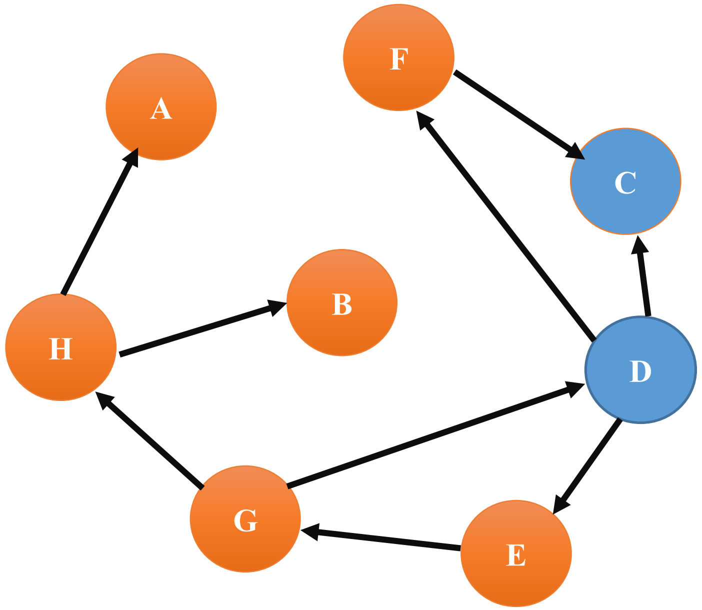
Ziyaret edilen

D

Yığın

Derinlik Öncelikli Arama (Depth First Search)

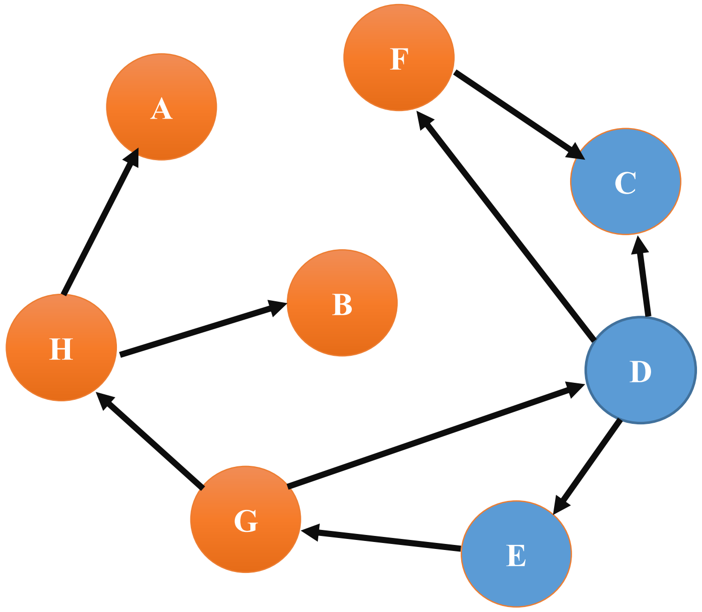
- C düğümünün komşusu olmadığı için (yönlendirilmiş çizge olduğuna dikkat ediniz) yığındaki en üstteki eleman ile devam edilecektir.



	Ziyaret edilen	Yığın
A		
B		
C	x	
D	x	
E		
F		C
G		E
H		F

Derinlik Öncelikli Arama (Depth First Search)

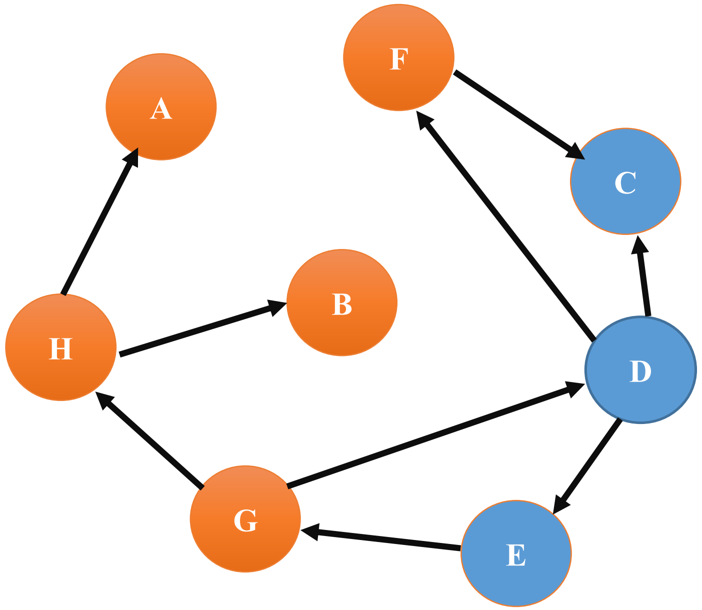
- C düğümünün komşusu olmadığı için (yönlendirilmiş çizge olduğuna dikkat ediniz) yığındaki en üstteki eleman ile devam edilecektir.



Ziyaret edilen		Yığın
A		
B		
C	x	
D	x	
E	X	
F		
G		E
H		F

Derinlik Öncelikli Arama (Depth First Search)

- E düğümünün komşuları yığına eklenir ve en üstteki seçilerek devam edilir



A	
B	
C	x
D	x
E	X
F	
G	
H	

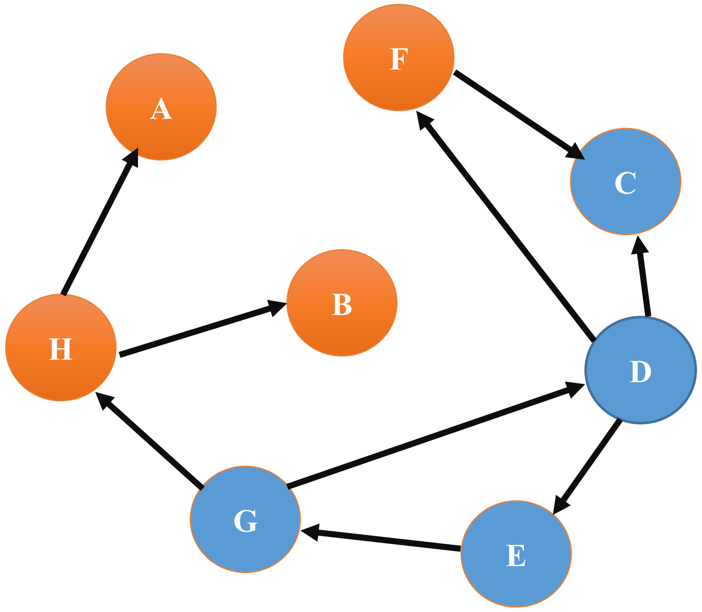
Ziyaret edilen

G
F

Yığın

Derinlik Öncelikli Arama (Depth First Search)

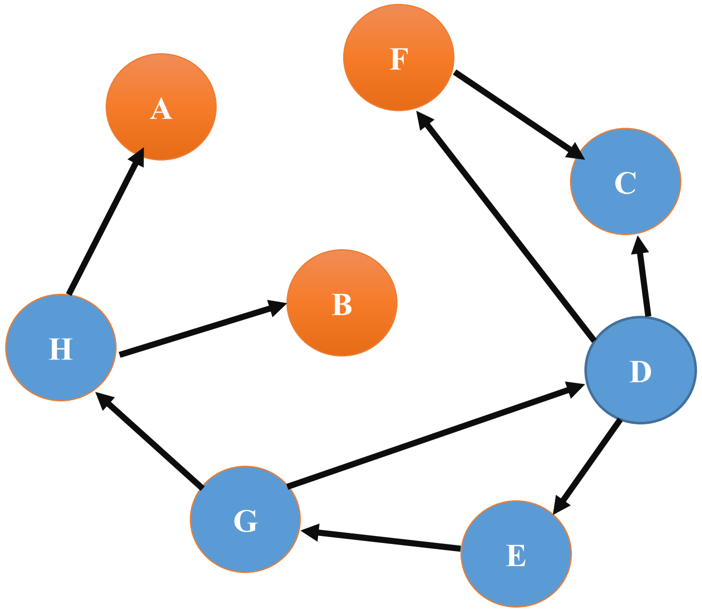
- G düğümünün komşuları yığına eklenir (D ziyaret edildiği için eklenmez) ve en üstteki seçilerek devam edilir.



	Ziyaret edilen	Yığın
A		
B		
C	x	
D	x	
E	X	
F		
G	X	H
H		F

Derinlik Öncelikli Arama (Depth First Search)

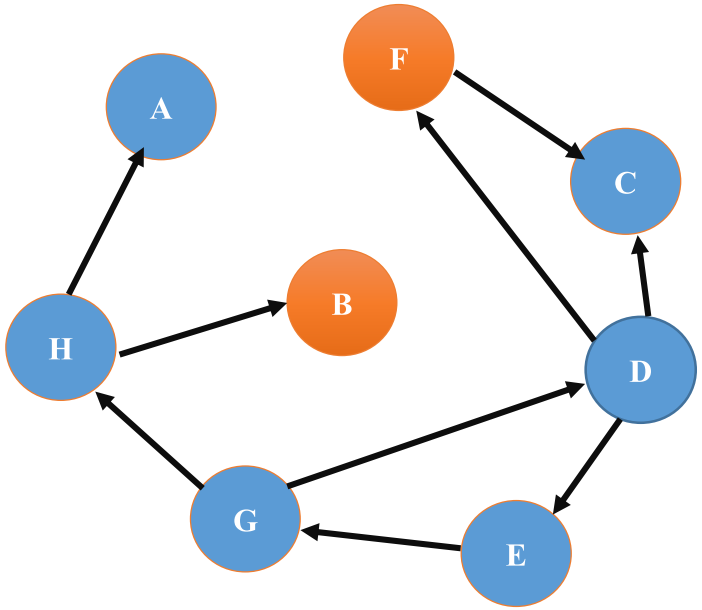
- H düğümünün komşuları yığına eklenir ve en üstteki seçilerek devam edilir.



Ziyaret edilen		Yığın
A		
B		
C	x	
D	x	
E	X	
F		A
G	X	B
H	X	F

Derinlik Öncelikli Arama (Depth First Search)

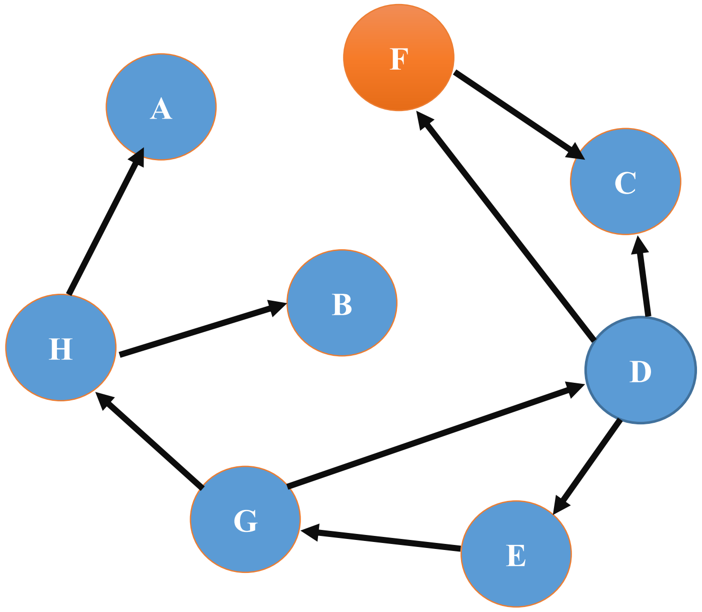
- A düğümünün komşusu olmadığı için (yönlendirilmiş çizge olduğuna dikkat ediniz) yığındaki en üstteki eleman ile devam edilecektir.



Ziyaret edilen		Yığın
A	X	
B	X	
C	x	
D	x	
E	X	
F		
G	X	B
H	X	F

Derinlik Öncelikli Arama (Depth First Search)

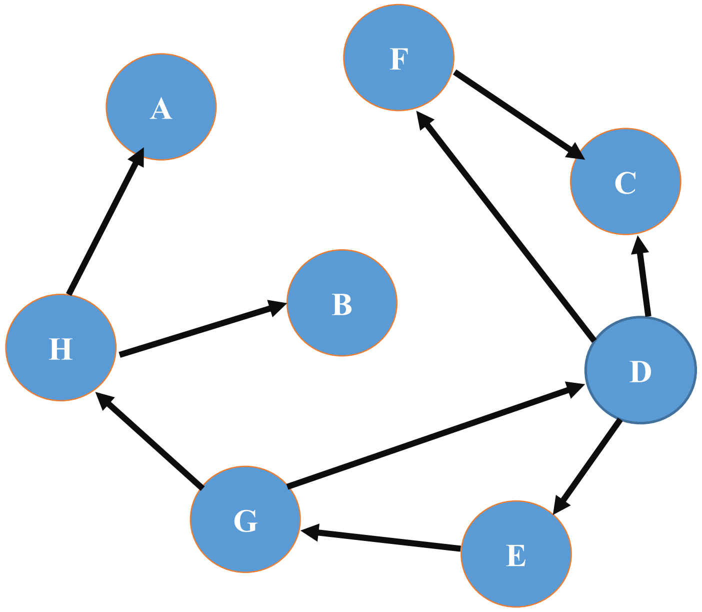
- B düğümünün komşusu olmadığı için (yönlendirilmiş çizge olduğuna dikkat ediniz) yığındaki en üstteki eleman ile devam edilecektir.



Ziyaret edilen		Yığın
A	X	
B	X	
C	x	
D	x	
E	X	
F		
G	X	
H	X	F

Derinlik Öncelikli Arama (Depth First Search)

- F düğümü de ziyaret edildikten sonra yığın boş kalacağı için algoritma sonlanacaktır.



A	X
B	X
C	x
D	x
E	X
F	X
G	X
H	X

Ziyaret edilen

Yığın

Derinlik öncelikli arama algoritması çalışma zamanı analizi:

- Derinlik öncelikli arama algoritması genişlik öncelikli arama algoritması ile benzerlik gösterir.
- Algoritmada her düğüm bir kez ziyaret edilecektir. Bu O(V) zamanda tamamlanır.
- İşlenen her düğümün tüm kenarı (u, v) için işlem yapılacağı için O(E) zaman gerekir.
- Bu nedenle derinlik öncelikli arama algoritmasının toplam çalışma zamanı O(V+E) olarak belirlenir.

En Küçük Kapsayan Ağaç Problemi

- Kapsayan ağaç, bir çizge üzerindeki tüm düğümleri içeren ve $düğüm\_sayısı - 1$ kenardan oluşan alt çizgedir.
- Tüm düğümleri içermesi nedeniyle kapsayan ağaç olarak adlandırılır. Kenarlar çift yönlü bağlantıları gösterdiği ve döngü içermediği için bu ağaçlarda herhangi iki düğüm arasında sadece tek bir yol bulunur.
- Bir çizge üzerinde birden çok kapsayan ağaç olabilir. En az maliyetli olan en küçük kapsayan ağaç (minimum spanning tree) olarak adlandırılır.

En Küçük Kapsayan Ağaç Problemi

- **Prim Algoritması:** En az maliyetli kenardan başlayıp onun uçlarından en az maliyetle genişleyecek kenarın seçilmesine dayanır. İşlem sırasında sürekli tek bir tane ağaç bulunur.
- **Kruskal Algoritması:** Daha az maliyetli kenarları tek tek değerlendirerek yol ağacını bulmaya çalışır. İşlem sırasında birden çok ağaç oluştur, sonunda bu ağaçlar birleşerek en küçük kapsayan ağacı oluşturur.

Prim Algoritması

- En az maliyetli kenardan başlayıp onun uçlarından en az maliyetle genişleyecek kenarın seçilmesine dayanır
- İşlem sırasında sürekli tek bir tane ağaç oluşur.
- Açgözlü bir yaklaşım izleyer
- Sürekli lokaldeki en düşük ağırlıklı kenarı seçerek globaldeki en düşük ağırlığa sahip ağacı oluşturur
- Prim algoritmasının basit bir sözde kodu aşağıda verilmiştir.

Prim Algoritması

Prim (G)

Adım 1. $V_T \leftarrow \{v_0\}$

Adım 2. $E_T \leftarrow \emptyset$

Adım 3. **for** $i \leftarrow 1$ **to** $|V| - 1$ **do**

Adım 4. Bütün (v, u) kenarları arasında v düğümü V_T içinde u düğümü de $V - V_T$ içinde olan en küçük ağırlığa sahip $e^* = (v^*, u^*)$ kenarını bul

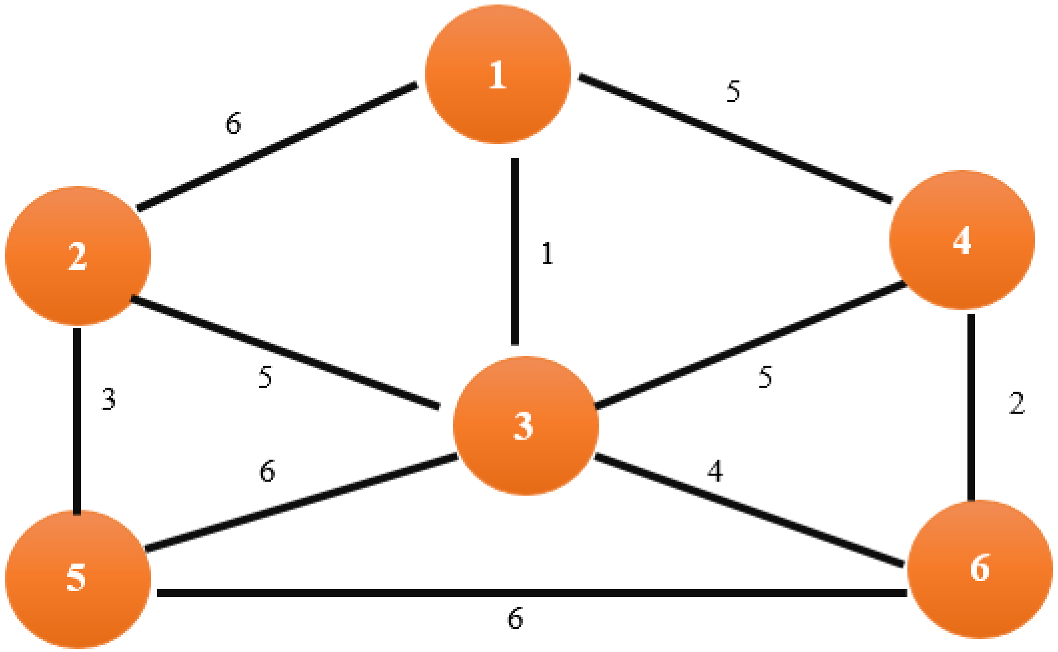
Adım 5. $V_T \leftarrow V_T \cup \{u^*\}$

Adım 6. $E_T \leftarrow E_T \cup \{e^*\}$

Adım 7. **return** E_T

Prim Algoritması

- 1 numaralı düğümden başlayarak en küçük kapsayan ağacı oluşturunuz.



Prim Algoritması

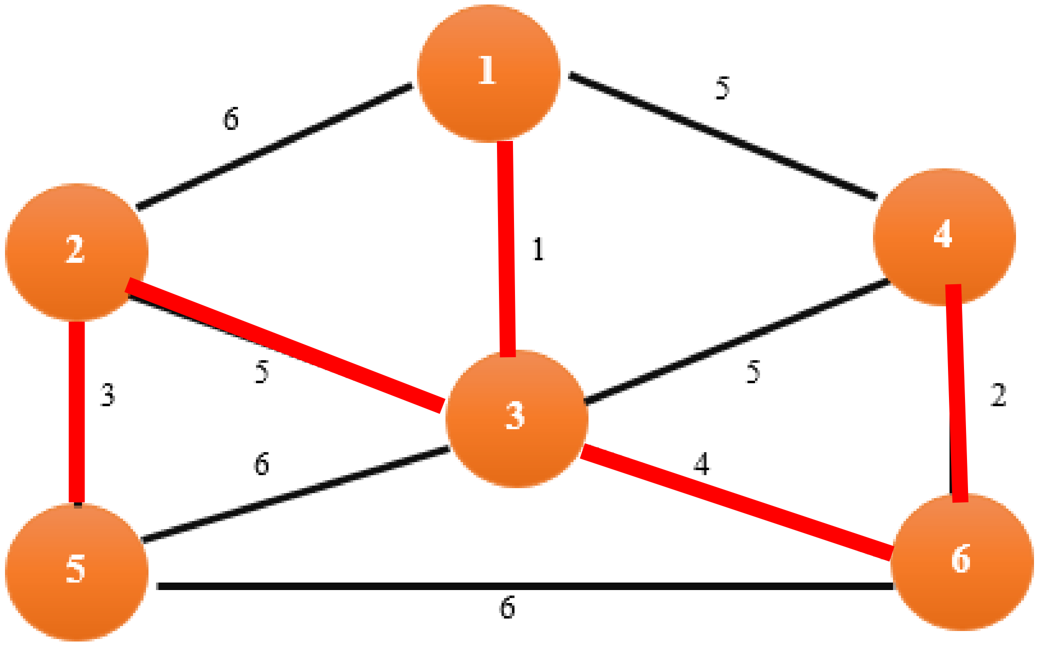
- Algoritma kapsayan ağaca dahil edilen düğümler ve geri kalan düğümler olmak üzere iki düğüm kümesi (set) tutar.
- Her aşamada en düşük ağırlığa sahip kenar seçilerek yeni bir düğüm ağaca katılır.
- 4. Adımdaki en düşük ağırlığa sahip kenarı bulabilmek için min-heap yada priority-queue kullanılabilir.
- Ayrıca bu adımda döngü kontrolünün de yapıldığına dikkat ediniz.

Prim Algoritması

- Her kenar için min-heap’den en düşük ağırlıklı kenarın bulunması O(lg$V$) zaman alır
- Algoritmanın toplam çalışma zamanı $O(E\lg V)$ değerine eşit olur.
- Aşağıda algoritmanın çalışmasını gösteren bir örnek verilmiştir.

Prim Algoritması

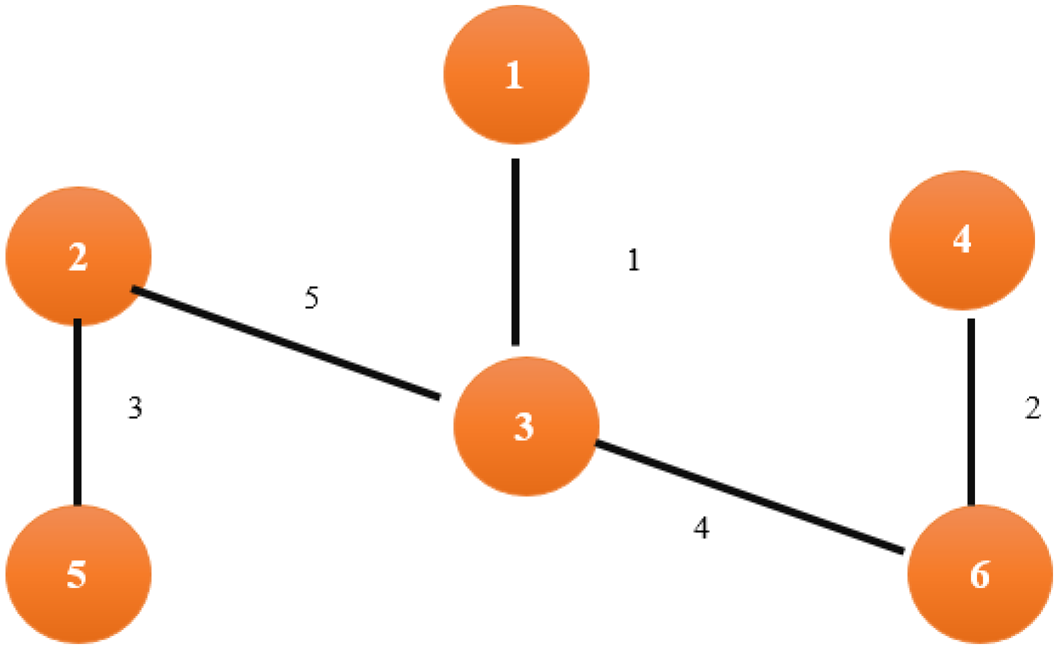
- 1 numaralı düğümden başlayarak en küçük kapsayan ağacı oluşturunuz.
- $V_T = \{1\}$
- $V_T = \{1,3\}$
- $V_T = \{1,3,6\}$
- $V_T = \{1,3,6,4\}$
- $V_T = \{1,3,6,4,2\}$
- $V_T = \{1,3,6,4,2,5\}$



Prim Algoritması

Adım 5.

$V_T = \{1,3,6,4,2\}$



Kruskal Algoritması

- Bu algoritma çizge üzerindeki düğümleri, aralarında bağlantı olmayan *n* tane bağımsız küme gibi düşünür.
- Prim’in algoritmasında olduğu gibi her seferinde en iyi (düşük maliyetli) kenarın seçilmesi esasına dayalıdır.
- *n* kenarlı bir çizge için herhangi bir düğümle başlanır ve düğüme bağlı en kısa yol çözüm kümesine eklenir.
- Döngü oluşturmayacak şekilde (*n*-1) kenar çözüm kümesine eklenene kadar devam edilir.
- Eğer aynı değere sahip birden çok kenar varsa herhangi bir tanesi seçilebilir.
- Algoritma sonunda tek bir ağaç elde edilmiş olur.

Kruskal Algoritması

- Algoritmaya ait sözde kod aşağıda verilmiştir.
- Kruskal algoritmasında başlangıçta çizge üzerindeki tüm kenarların küçükten büyüğe sıralanmış olması gerekir.
- Sözde kodda kullanılan E_T dizisi (çözüm kümesi) çizgedeki işlenen kenarları küçükten büyüğe sıralı bir şekilde tutmaktadır.
- Algoritma her aşamada E_T'ye ekleyeceği kenarın döngü oluşturup oluşturmadığını kontrol ederek ekler.

Kruskal Algoritması

Kruskal (G)

Adım 1. $E_T \leftarrow \emptyset$

Adım 2. $kenar \leftarrow 0$ // çizge boyutu için sayaç

Adım 3. $k \leftarrow 0$ // işlenen düğümler için sayaç

Adım 4. **while** $kenar < |V| - 1$ **do**

Adım 5. $\quad k \leftarrow k + 1$

Adım 6. $\quad$ **if** $E_T \cup \{e_{i_k}\}$ döngü oluşturmuyorsa

Adım 7. $\qquad E_T \leftarrow E_T \cup \{e_{i_k}\}$

Adım 8. $\qquad kenar \leftarrow kenar + 1$

Adım 9. **return** E_T

Kruskal Algoritması

- Sözde koddan anlaşılacağı üzere Kruskal algoritmasında çözüm kümesi düşük ağırlığa sahip küçük ağaçlardan oluşan bir ormandır.
- Çözüm kümesine her aşamada eklenen kenar her zaman çizge içinde ayrık iki parçayı (ağacı) birleştiren ve döngü oluşturmayan en düşük ağırlıklı kenardır.
- Burada algoritmanın aç gözlü bir yaklaşım izleyerek lokaldeki en düşük ağırlıklı kenarı seçerek globaldeki en düşük ağırlığa sahip ağacı oluşturduğuna dikkat ediniz.

Kruskal Algoritması

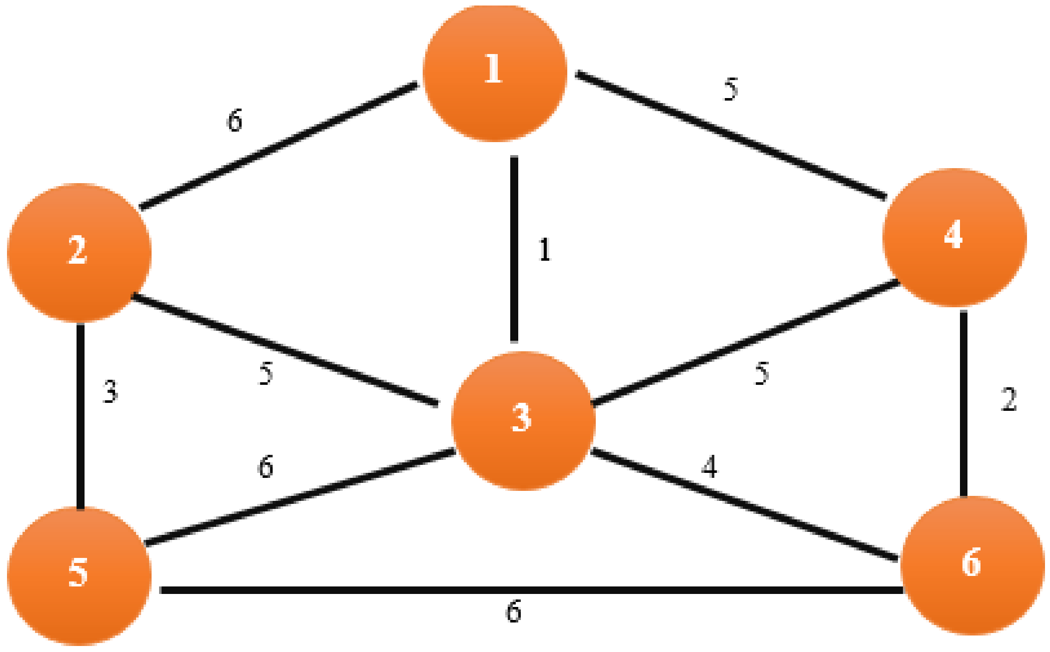
- Kruskal algoritmasının çalışma zamanı döngü kontrolünün nasıl yapıldığına göre değişir.
- En kolay kontrol yöntemi işlenen kenarlar ve işlenmemiş kenarlarda yer alan düğümler için iki küme (set) oluşturup eklenecek kenarın düğümlerinin bu setlerde olduğunun kontrol edilmesidir.
- (*u,v*) kenarı eklenecek ise *u* düğümün bulunduğu küme ve *v* düğümünün bulunduğu küme aynı değilse bu kenar döngü oluşturmuyor demektir.

Kruskal Algoritması

- Küme fonksiyonları ile 4. Adımdaki döngü kontrolü O(lg*E*) zaman içinde gerçekleştirilebilir,
- Bu işlem her kenar için yapılacağından Kruskal algoritmasının toplam çalışma zamanı O(*E*lg*E*) olur.
- Ancak çizge özelliklerinden $|E| < |V^2|$ olduğunu hatırlayınız, bu durumda her iki tarafında lg değerini alırsak lg*E*=O(*V*) olur.
- Buna göre Kruskal algoritmasının çalışma zamanı ifadesi O(*E*lg*V*) olarak yazılabilir.
- Bu çalışma zamanının Prim'in algoritmasına eşit olduğuna dikkat ediniz.

Kruskal Algoritması

- Aşağıda verilen çizgede en küçük kapsayan ağacı oluşturunuz



Kruskal Algoritması

- Sıralı kenarlar:
- {1,3}
- {4,6}
- {2,5}
- {3,6}
- {2,3}