VERI YAPILARI VE ALGORITMALAR

BLM2512 Gr.2

2020-2021 Bahar Yarıyılı (Uzaktan Eğitim)

Dr.Öğr.Üyesi Göksel Biricik

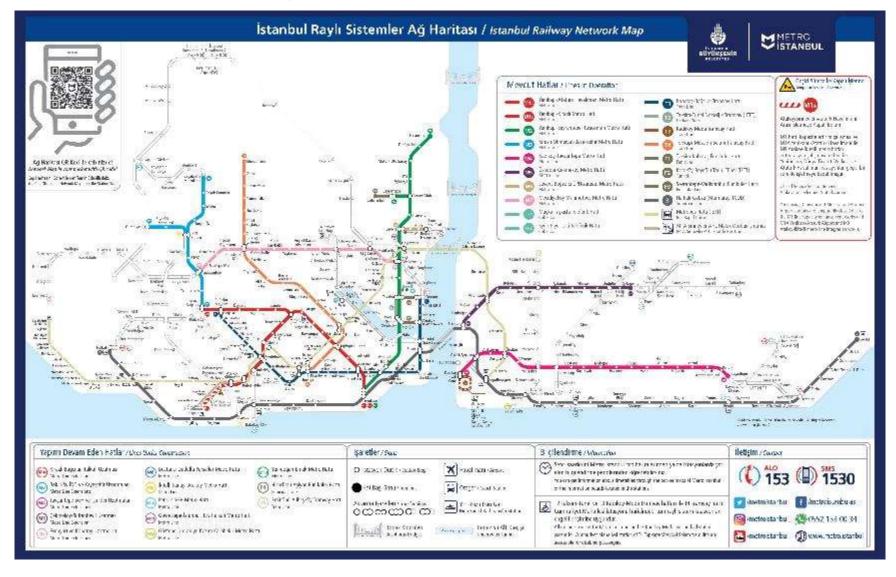
GRAFLAR

Çizgeler

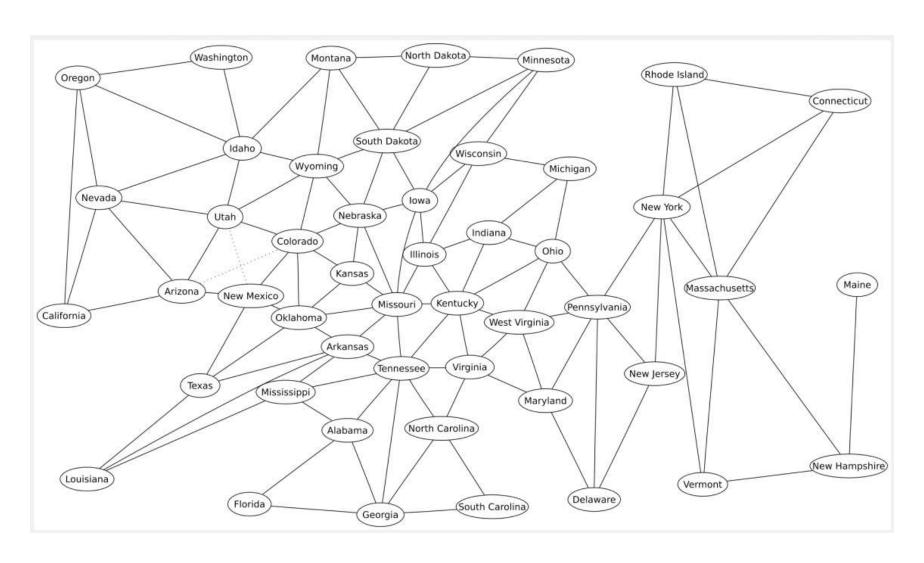
Graflar

- Birbirine bağlı elemanlar kümesine graf adı verilir.
- G(V,E)
- V (vertex) Düğüm, |V| Düğüm sayısı
- E (edge) Kenar, |E| Kenar sayısı
- Karmaşıklık gösteriminde, O(VE) → O(|V||E|)

İstanbul Raylı Sistemleri



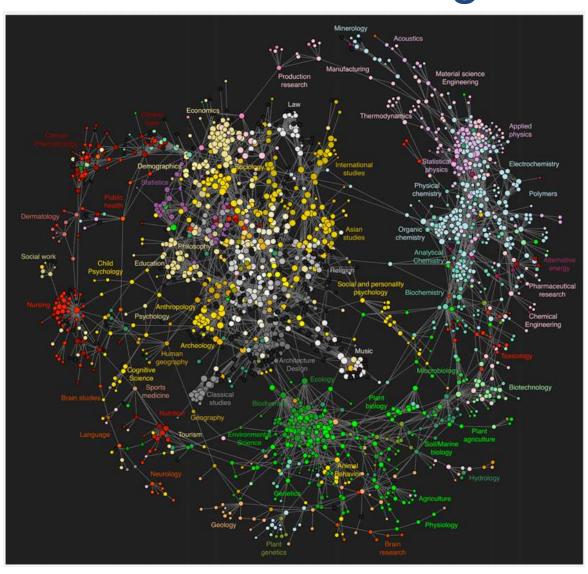
ABD Eyaletleri Sınır Komşulukları



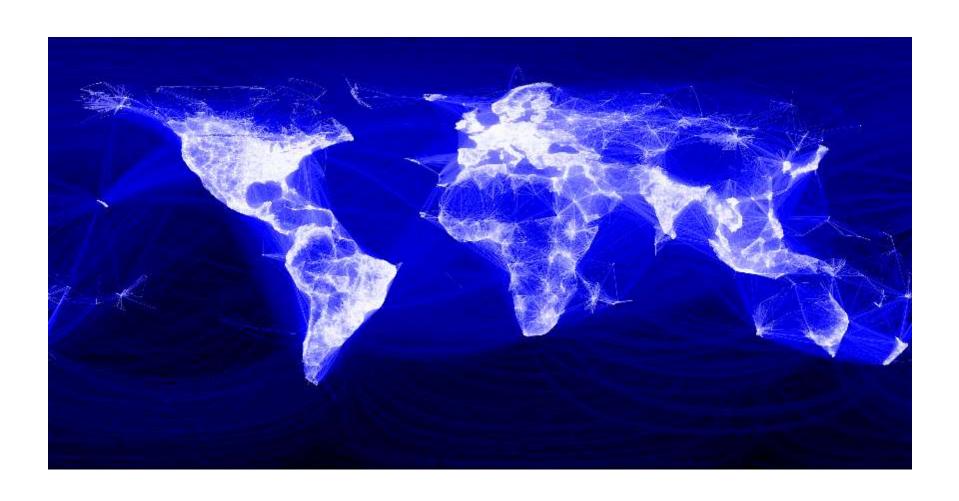
Protein-Protein Etkileşim Ağı



Bilim Konuları Tıklama Bağlantıları



FaceBook Arkadaşlık Bağlantıları

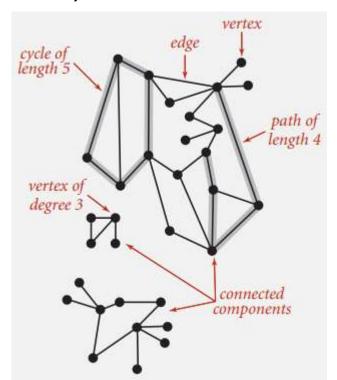


Graf Kullanım Alanları

- Haritalar:
 - Beşiktaş-Davutpaşa arası en kısa yol nedir? (Shortest Path)
 - Beşiktaş-Davutpaşa arası en hızlı yol nedir? (Max Flow)
- Web İçeriği: Arama Motorları
- Devreler: Kısa devre var mı? Bağlantılar çaprazlamadan gerçeklenebiliyor mu?
- Zaman Planlaması/Tarife: Birbiri ile bağlı işler sırası, kısıtlar altında en kısa sürede nasıl tamamlanır?
- Ticaret: Alıcı-Satıcı-Ürün Ağı
- Eşleştirme: Öğrenci Kulüp/Staj eşleştirme
- Bilgisayar Ağları
- Yazılım: Derleyicilerin modüller arası statik/dinamik çağrıları, kaynakları modellemesi
- Sosyal ağlar: Anomali, kanaat önderi, bot ağı...

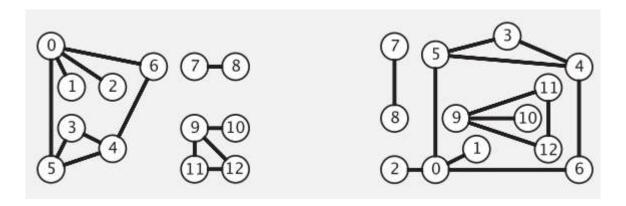
Graf Terminolojisi

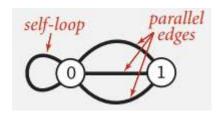
- Path (Yol): Kenarlarla bağlanmış olan düğümler silsilesi.
- Cycle (Çevrim): İlk ve Son düğümü aynı olan yollar.
- İki düğüm arasında bir yol varsa, bu iki düğüm birbirine bağlıdır (connected).



Graf Tipleri

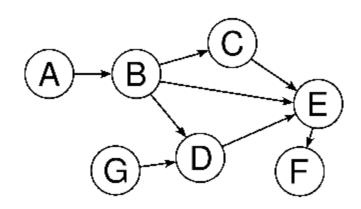
- Yönsüz Graf (Undirected)
- Yönlü Graf (Digraph)
- Kenar Ağırlıklı Yönsüz Graf
- Kenar Ağırlıklı Yönlü Graf

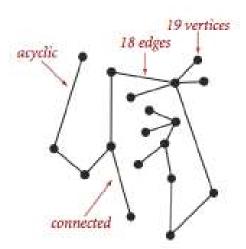




Acyclic (Çevrimsiz) Graf

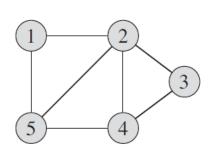
- Kapalı Döngü içermeyen graflar.
- 5 şart doğru ise ağaçtır:
 - V-1 kenar var, döngü yok
 - V-1 kenar var, bileşenleri bağlı
 - Herhangi bir kenarı kaldırmak, ağacı keser
 - Çevrimsizdir, bir kenar eklemek cyclic yapar
 - G'nin her kenar çiftini basit bir yol bağlar.

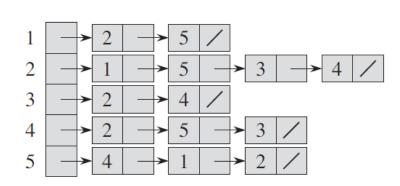




Graf Gösterimi (Graph Representation)

- Komşuluk Listesi (Adjacency List)
 - Seyrek (Sparse) graflar için uygun
- Komşuluk Matrisi (Adjacency Matrix)
 - V² yer kaplar
 - $A_{ij} = 1$, if $i,j \in E$; 0 otherwise
 - \bullet $A^T = A$
 - 1 bit ile tutabiliriz.

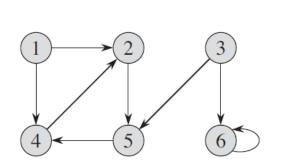


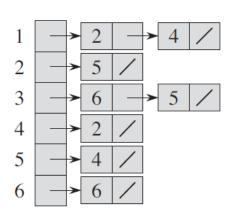


	1	2	3	4	5
1	0.	1	0	0	1
2	1	0.	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1 1 1	0	1	0.

Graf Gösterimi (Graph Representation)

Yönlü grafta A^T != A





	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	1 0 0 0 0 0	0	0	0	0	1

- Ağırlıklı (weighted) graflarda matriste değerler yer alır.
- Bağlantı var mı?
 - Listede yavaş O(degree(V))
 - Matriste hızlı O(1)

GRAF ARAMA YÖNTEMLERİ

BFS, DFS

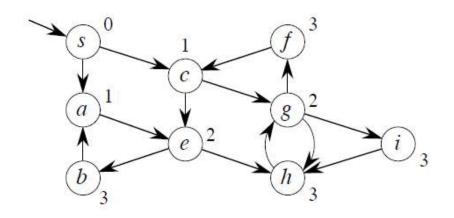
Önce Genişlemesine Arama (Breadth First Search, BFS)

- En basit arama yöntemlerinden biridir.
- Pek çok algoritma BFS'ye benzer mantık kullanır.
 - Prim MST, Dijkstra SP
- Girdi: G(V,E), yönlü ya da yönsüz, kaynak düğüm S∈V
- S'den ulaşılabilen tüm düğümleri «keşfetmeye» çalışır.
 - «Breadth-First Tree» (S kökünden erişilebilen tüm yollar) yaratır.
 - S'den erişilebilen v düğümlerine giden en kısa yolları içerir.
- K+1 uzaklığa geçmeden önce, K uzaklıktaki tüm yollar bulunur.

Önce Genişlemesine Arama (Breadth First Search, BFS)

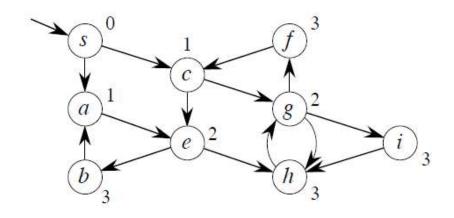
- Fikir:
 - S'ten bir dalga gönder.
 - İlk önce S'ten 1 uzaklıktakilere çarpar.
 - Onlardan, 2 uzaklıktakilere çarpar.
 - ...
- Dalganın önünde kim olacak?
- FIFO queue
 - Dalga v'ye çarptı ve daha v'yi terketmediyse, v ∈ Q.

```
BFS(V, E, s)
for each u \in V - \{s\}
  do d[u]←∞
d[s] \leftarrow 0
Q \leftarrow \emptyset
ENQUEUE(Q, s)
while Q != Ø
  do u \leftarrow \mathsf{DEQUEUE}(Q)
    for each v \in Adj[u]
       do if d[v] = \infty
         then d[v] \leftarrow d[u] + 1
                ENQUEUE(Q, v)
```



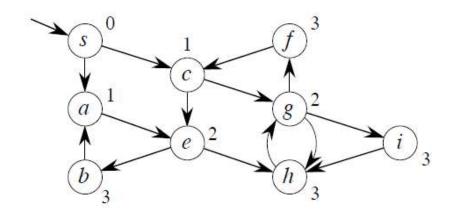
• Q = *NULL*

	a	b	C	е	f	g	h	i	S	
d	∞	∞	∞	∞	∞	∞	∞	∞	0	



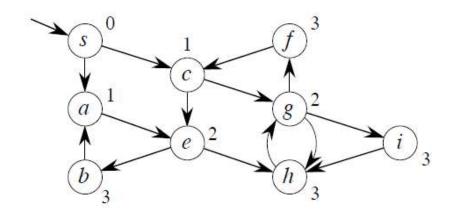
- $Q = \{s\}$
- u=s [a,c] d[a]=d[s]+1, d[c]=d[s]+1
- Q={a,c}

	а	b	С	е	f	g	h	i	S
d	1	∞	1	∞	∞	∞	∞	∞	0



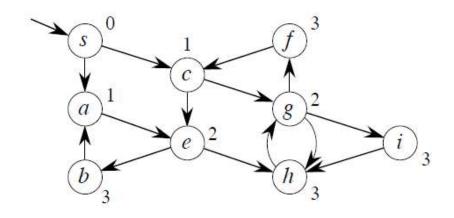
- $Q = \{a, c\}$
- u=a [e] d[e]=d[a]+1
- Q={c,e}

	а	b	С	е	f	g	h	i	S
d	1	∞	1	2	∞	∞	∞	∞	0



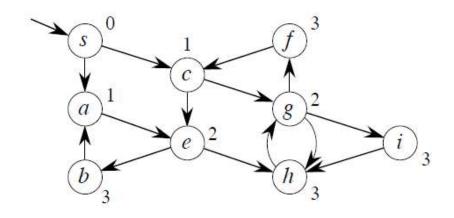
- $Q = \{c, e\}$
- u=c [e,g] d[e]!=∞, d[g]=d[c]+1
- Q={e,g}

	а	b	С	е	f	g	h	i	S
d	1	∞	1	2	∞	2	∞	∞	0



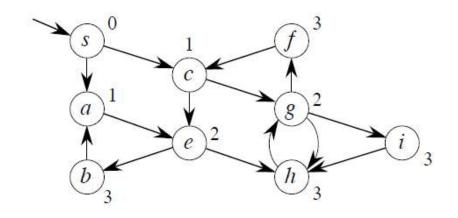
- $Q = \{e,g\}$
- u=e [b,h] d[b]=d[e]+1, d[h]=d[e]+1
- Q={*g*,*b*,*h*}

	a	b	С	е	f	g	h	i	S
d	1	3	1	2	∞	2	3	∞	0



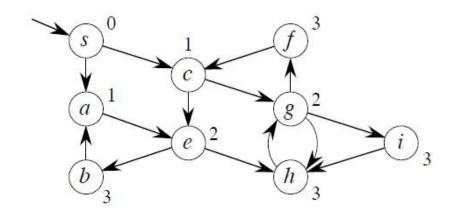
- $Q = \{g, b, h\}$
- u=g [f,h,i] d[f]=d[g]+1, d[h]!=∞, d[i]=d[g]+1
- Q={b,h,f,i}

	а	b	С	е	f	g	h	i	S
d	1	3	1	2	3	2	3	3	0



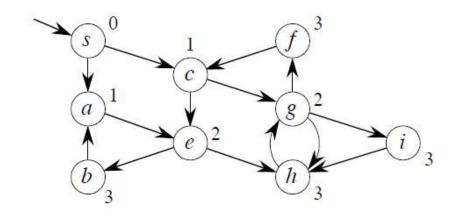
- $Q = \{b, h, f, i\}$
- u=b [a] d[a]!=∞
- Q={h,f,i}

	а	b	С	е	f	g	h	i	S
d	1	3	1	2	3	2	3	3	0



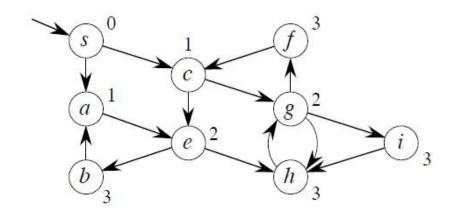
- $Q = \{h, f, i\}$
- u=h [g] d[g]!=∞
- Q={f,i}

	а	b	С	е	f	g	h	i	S
d	1	3	1	2	3	2	3	3	0



- Q = $\{f, i\}$
- u=f [c] d[c]!=∞
- Q={*i*}

	а	b	С	е	f	g	h	i	S
d	1	3	1	2	3	2	3	3	0



- $Q = \{i\}$
- u=i [h] d[h]!=∞
- Q={} → stop

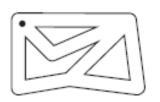
	а	b	С	е	f	g	h	i	S
d	1	3	1	2	3	2	3	3	0

BFS Karmaşıklık Analizi

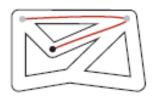
- Enqueue / Dequeue O(1)
- Her düğüm en fazla 1 kere enqueue, toplam O(V)
- Her düğüm en fazla 1 kere dequeue ve o zaman (u,v) kontrolü: O(E)
- O(V+E) (+ init O(V))
- Komşuluk listesi gösterimi boyutunda lineer zamanda çalışır.

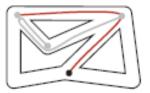
Tremaux Exploration

- intersection
- Bir labirentin girişindeyiz. Elimizde bir yumak
 ip var. Kaybolmadan labirentten nasıl çıkarız?
- 1. İşaretsiz bir geçide ip döşe
- 2. İlk kez geçtiğin tüm geçit ve kesişimleri işaretle
- 3. İşaretli bir kesişime gelirsen, ip ile geri gel
- Geri gelirken hiçbir işaretsiz kesişim kalmayana kadar adımları geri al.













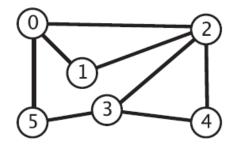
DFS de bu yönteme benzer. Hatta daha kolaydır ©

Önce Derinlemesine Arama (Depth First Search, DFS)

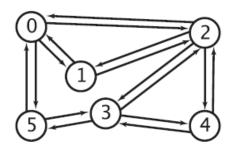
- Düğümleri rekürsif olarak ziyaret et.
- Bir düğümü al, işaretle.
- Tüm (işaretsiz) komşularını rekürsif olarak ziyaret et.

```
DepthFirstSearch(G,s)
 count=0;
 for each vertex u \in G.V
   marked[u]=FALSE
 dfs(G,s)
dfs(G,v)
 marked[v]=TRUE
 count++
 for each w \in G.adj[v]
   if (! marked[w])
     dfs(G,w)
```

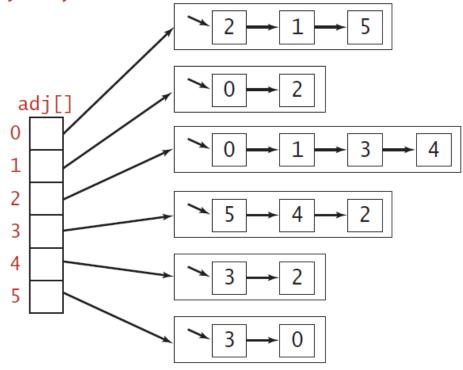
standard drawing



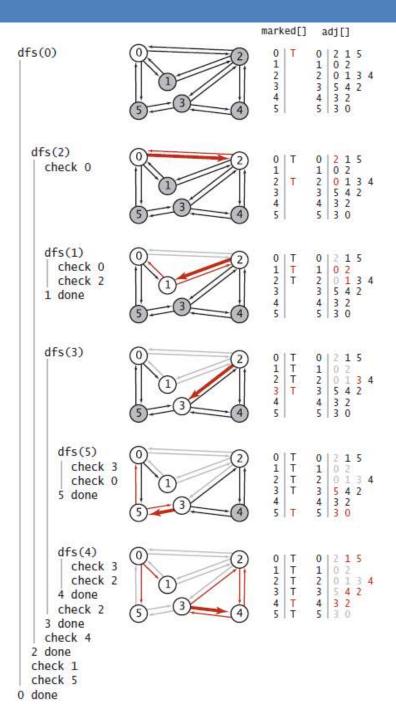
drawing with both edges







- 2, 0'ın ilk komşusu ve unmarked. Visit 2.
- 2'nin ilk komşusu 0 ve marked. Geç.
 Sıradaki komşu 1 ve unmarked. Rekürsif olarak işaretle ve visit 1.
- 1'in tüm komşuları [0,2] marked. Geri dön. 2'nin sıradaki komşusu 3'ü işaretle ve visit (unmarked).
- 3'ün ilk komşusu 5 ve unmarked. İşaretle ve Visit 5.
- 5'in tüm komşuları [3,0] marked. Geri dön. 3'ün sıradaki komşusu 4'ü işaretle ve visit (unmarked).
- 4'ün komşularını kontrol et, geri dön 3'ün kalan komşularını kontrol et, geri dön 2'nin kalan komşularını kontrol et, geri dön 0'ın kalan komşularını kontrol et.
- Tüm gezinti bitti.

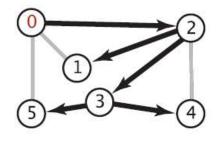


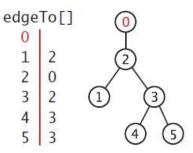
DFS ile neyi çözebiliriz?

```
Tek kaynaktan nerelere gidebiliriz?

    Verilen 2 düğüm birbirine bağlı mıdır? ⇔ 2 düğüm arası yol var mıdır?

DepthFirstPaths(G,s)
  for each vertex u € G.V
   marked[u]=FALSE
   edgeTo[u] = 0;
  dfs(G,s)
dfs(G,v)
 marked[v]=TRUE
 for each w € G.adj[v]
   if (! marked[w])
     edgeTo[w]=v
     dfs(G,w)
pathTo(v)
 // path is a stack
 if (! marked[v]) return NULL
 for (x=v; x!=s; x=edgeTo[x])
   path.push(x)
  path.push(s)
  return path
```





X	path					
5	5					
3	3	5				
2	2	3	5			
0	0	2	3	5		

MINIMUM SPANNING TREE

Asgari Tarama/Örtme Ağacı

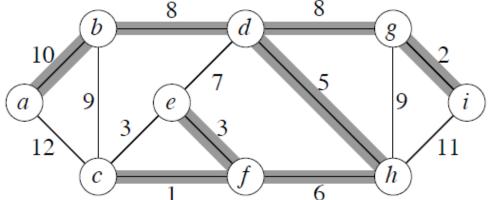
Minimum Spanning Tree

- Bir köyün muhtarısınız.
- Sorumluluğunuz gereği, köydeki tüm evleri birbirine bağlamanız gerek.
 - Bir yol parçası w ile 2 ev (u, v) bağlanabilir.
 - Bu yol parçasının yapım/tamir masrafı w(u,v)'dir.
- Hedefiniz,
 - Herkesi birbirine bağlı tutacak (bir evden diğer tüm evlere ulaşılabilecek)
 - Bunu minimum masraf ile yapacak

Yol ağını oluşturmaktır.

Minimum Spanning Tree

- Çözüm: Köyünüzü yönsüz graf olarak modelleyin. G=(V,E)
- Her (u,v) € E kenarı için bir ağırlık w(u,v) atayın.
- Öyle bir $T \subseteq E$ bulun ki;
 - T tüm düğümleri birbirine bağlasın (tarasın/örtsün)
 - $w(T) = \sum_{(u,v) \in T} w(u,v)$ Değeri asgaride kalsın.
- Tüm örten ağaçlar içerisinde ağırlıklarının toplamı en az olan ağaca asgari tarama ağacı (minimum spanning tree) adı verilir.



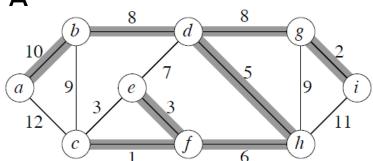
Minimum Spanning Tree

- G(V,E) grafımızda tanımlı w:E $\rightarrow \mathbb{R}$ ağırlık fonksiyonumuz olsun.
- G için tanımlı bir MST (min w) bulmak istiyoruz.
 - MST'nin |V|-1 kenarı olmalı. ←
 - · Çevrim (cycle) içermemeli.
 - G için birden fazla MST bulunabilir.
- Kenarlardan oluşan bir A kümesi oluşturmalıyız.
- Boş küme ile başlayıp, A'ya kenar ekleyerek büyütürüz.
 - Döngü şartı: A, bir MST'nin alt kumesi olmalıdır.
- Bu durumda ancak güvenli kenarları kümeye eklemeliyiz.
- Her adımda bir stratejiye göre davranıp eylem gerçekleştiriyoruz. → Greedy çözümler MST oluşturmak için uygun.

Generic MST Algoritması

GENERIC-MST(G,w)

- $1 A = \emptyset$;
- 2 while A does not form a spanning tree
- 3 find an edge (u,v) that is safe for A
- 4 $A = A \cup \{(u,v)\}$
- 5 return A
- Güvenli kenar nasıl bulunacak?
 - (c,f) kenarı en düşük maliyetli olan. A için güvenli mi?
 - O ana kadar oluşan ağaçta c ∈ S olsun ama f düğümü yok (f ∈ V-S)
 - Herhangi bir MST'de en az bir kenar ile f de ağaca bağlanmalıdır.
 Neden en ucuz olanını seçmeyelim ki? (Greedy seçim stratejisi)



Generic MST Algoritması

- S ve V-S ayrık kısımlarına kesim (cut) diyelim.
- (S, V-S) kesimleri arasında kenarlar (u,v) € E olabilir.

 Bu kesimler arasında bağlantıyı hafifleten bir kenar bizim için güyenlidir.

V-S

için güvenlidir.

 Örnekte, (u,v) kenarının değeri (x,y) kenarının değerinden küçük olsun. (x,y)'yi kaldırıp (u,v)'yi ekleriz, Spanning tree yapısını bozmamış oluruz.

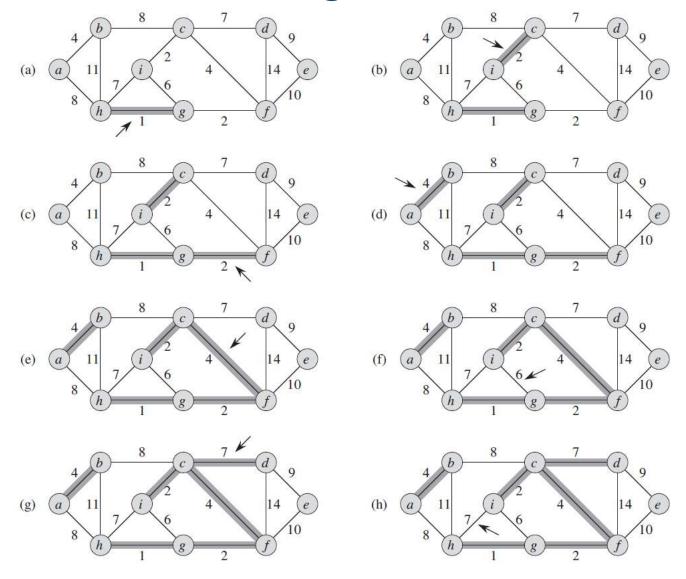
Generic MST Algoritması

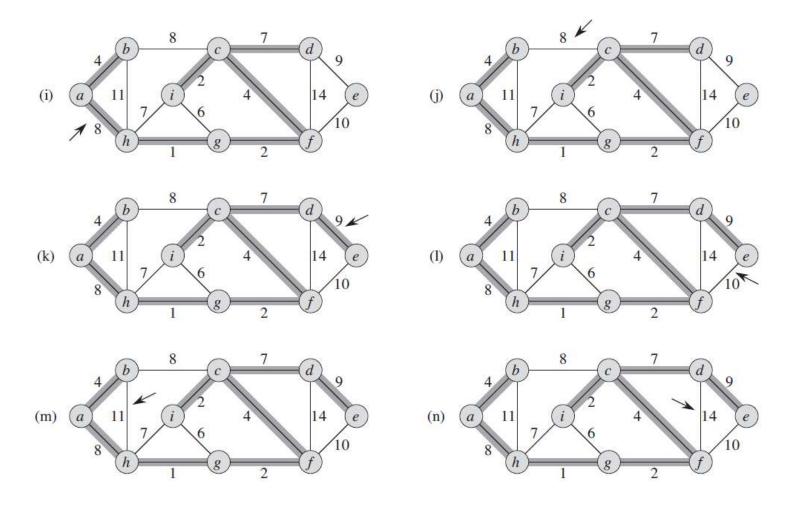
- Algoritmadaki A kümesi, bağlı bileşenlerden oluşan bir ormandır.
 - Başlangıçta, her bileşen tek bir düğümdür.
- Herhangi bir güvenli kenar, iki bileşeni birleştirerek tek bileşen haline getirir.
 - Her bileşen bir ağaçtır.
- MST'de |V|-1 kenar olduğundan, döngü |V|-1 kere döner.
 - |V|-1 güvenli kenarı eklediğimizde, elimizde tek bir bileşen oluşur.
- Bu noktadan devam edersek, Kruskal'ın MST algoritma çözümüne ulaşırız.

KRUSKAL MST

- Her düğüm bir bileşen olacak şekilde başlar.
- Tekrarlı olarak, iki bileşeni birbirine bağlayacak «hafif» bir kenar bularak devam eder (kesimler arasındaki hafif kenar).
- Kenarları, ağırlıklarına göre artan şekilde tarar.
- Bir kenarın farklı bileşenlerdeki düğümleri bağlayıp bağlamadığını belirlemek için ayrık küme şeklinde veri yapısı kullanır.

```
KRUSKAL(V, E,w)
A \leftarrow \emptyset
for each vertex v \in V
  do MAKE-SET(v)
sort E into nondecreasing order by weight w
for each (u, v) taken from the sorted list
  do if FIND-SET(u) = FIND-SET(v)
       then A \leftarrow A \cup \{(u, v)\}
             UNION(u, v)
return A
```





Kruskal MST Karmaşıklık Analizi

- Init A: O(1)
- İlk for döngüsü: |V| (MAKE_SET)
- E'ye göre sıralama: O(E IgE)
- İkinci for döngüsü: O(E) FIND_SET ve UNION
- O(E Ig V)
- Eğer kenarlar sıralıysa, O(E a(V)), neredeyse lineer

PRIM MST

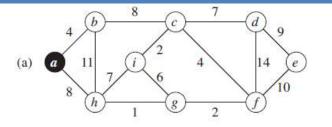
Prim'in MST Algoritması

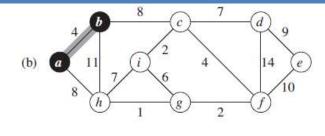
- Tek bir ağaç oluşturur.
 - A her zaman bir ağaçtır.
- Keyfi bir r kökü ile başlar.
- Her adımda, kesimler arasında bir hafif kenar bularak ağaca ekler. (V_A, V-V_A)
- V_A
- Hafif kenar nasıl bulunabilir? Öncelikli Kuyruk ile.
 - Kuyruktaki elemanlar V-V_A kümesindeki düğümlerdir.
 - Elemanların anahtarları, ağırlık değerleridir.
 - EXTRACT_MIN ile (V_A, V-V_A) arasındaki hafif kenar geçişi bulunur.
 - Eğer düğüm V_A'daki düğümler ile komşu değilse, ağırlığı sonsuzdur.
- A'nın kenarları, r köküne sahip bir ağaç oluşturacaktır.
 - r başlangıcı verilir ama esasen herhangi bir düğüm olabilir.
 - Her düğüm, ağaçtaki ebeveynini (p[v]) bilir.
 - Algoritma ilerledikçe V_A büyür, V boş küme olduğunda (tüm düğümlere erişildiğinde) sonlanır.

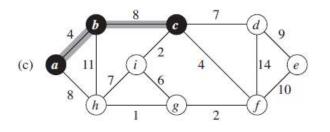
Prim'in MST Algoritması

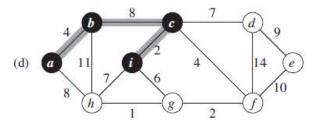
```
PRIM(V, E, w, r)
Q \leftarrow \emptyset
for each u \in V
  do key[u] \leftarrow \infty
      \pi[u] \leftarrow \mathsf{NIL}
      INSERT(Q, u)
DECREASE-KEY(Q, r, 0) key[r] \leftarrow 0
while Q != Ø
  do u \leftarrow \mathsf{EXTRACT}-MIN(Q)
      for each v \in Adj[u]
          do if v \in Q and w(u, v) < key[v]
                 then \pi[v] \leftarrow u
                        DECREASE-KEY(Q, v, w(u, v))
```

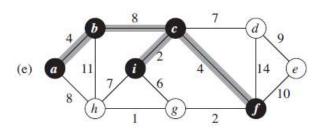
Prim'in MST Algoritması

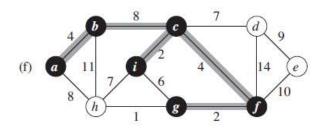


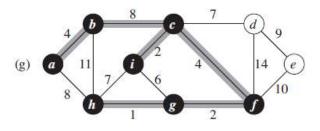


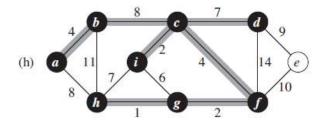


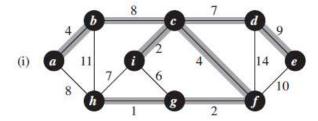












Prim MST Karmaşıklık Analizi

- Q binary heap ile yapılmış olsun.
- Init Q ve ilk for döngüsü: O(V lgV)
- R anahtar değerinin azaltılması: O(lgV)
- While döngüsü
 - |V| Extract_min çağrısı → O(V lgV)
 - <= |E| Decrease_key çağrısı → O(E IgV)
- O(E lg V)
- Eğer (fibonacci heap kullanarak) Decrease_key O(1)'de yapılırsa, O(V IgV + E)