

# Graphs (çizge

todo

## 6.1 hedefler

- çizge nedir? nasıl kullanılır?
- çoklu içsel temsille çizge SVT gerçekleştirilmesi nasıl?
- çizgeler, çok geniş problem ailesini çözümlemede kullanılabilir. nasıl?

# giriş

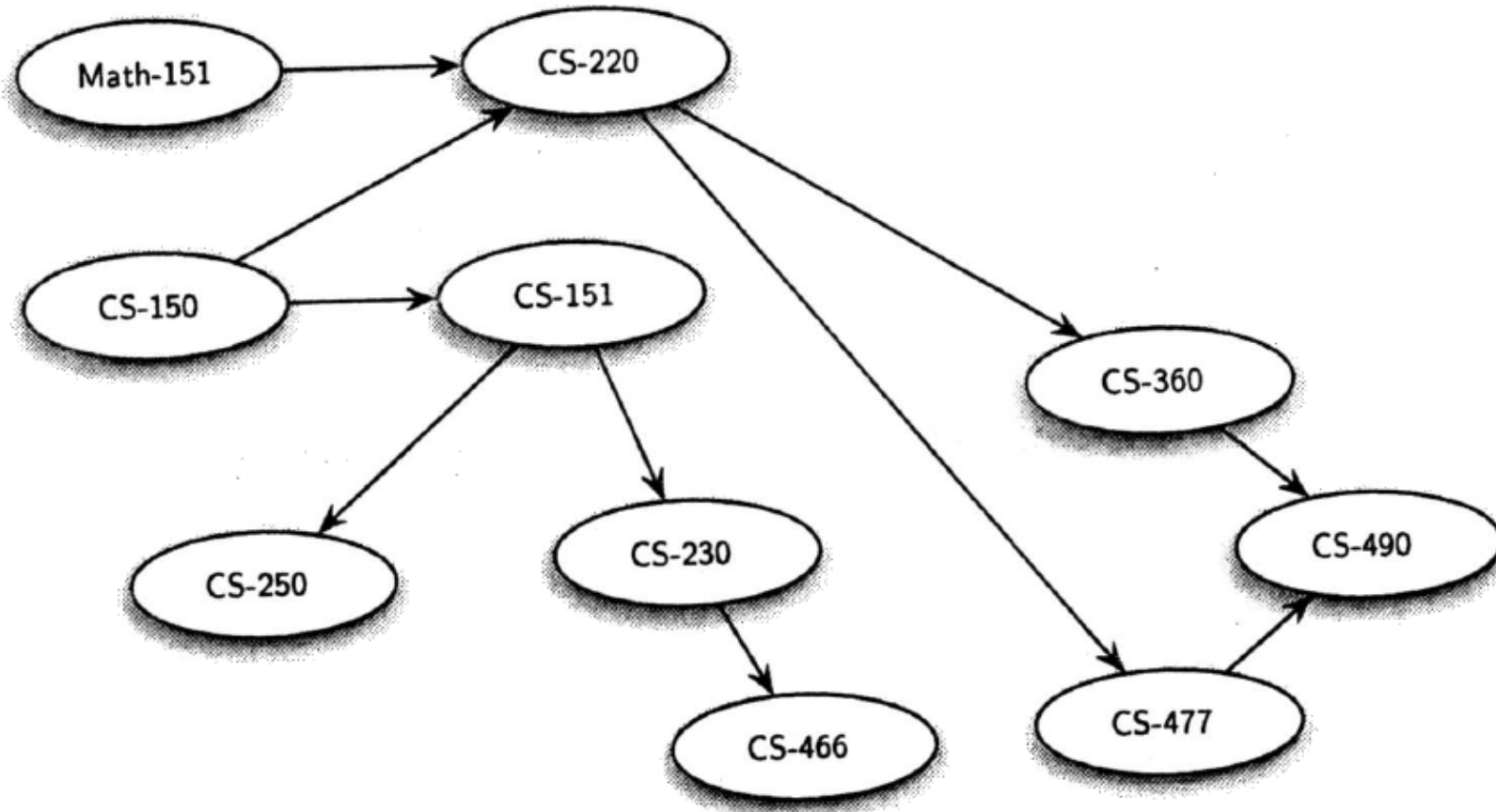
- çizge, ağaçlardan daha geneldir
- ağaç, çizgenin özel türüdür
- çizge: yol, hava yolu, Internet bağlantısı, ders ağacı
- çizgelerle çözümü kolay olan problemler --> çizge algoritmaları

# çizge algoritmaları

- bizler haritaya bakarak anlayabiliyor, kullanabiliyoruz
- bilgisayar bundan anlamıyor
- en kısa yol nedir?
- en hızlı nasıl ulaşırım
- en kolay yol hangisidir?
- benzeri sorularının karşılığı yoktur.
- fakat harita --> çizge dönüşümü
- çizge algoritmaları iş görür

# örnek

- önkoşul çizgesi
- dersler arası ilişki



**Figure 6.1:** Prerequisites for a Computer Science Major

## 6.2 sözlük ve tanımlar

vertex:

- düğüm, ismi var, “key” adlandırılır
- ek bilgi içerebilir, payload adlandırılır

edge:

- giriş, iki düğümü bağlar
- yön: tek/çift
- tüm girişler tek yönlüyse “directed graph”
- yönlü çizge ise “digraph”
- önkoşul çizgesi “digraph”

# sözlük ve tanımlar

weight:

- ağırlık - mesafe
- bir düğümden diğerine gitme maliyeti
- düğüm=şehir ise ağırlık=mesafe olabilir

## devam

→ çizgenin formal tanımı

$$G = (V, E)$$

→ G: graph, V: vertex, E: edge

→ çizge, düğüm ve kırışler kümesidir

→ her bir kırış bir tuple'dır

$$e = (v, w) \in V \times V$$

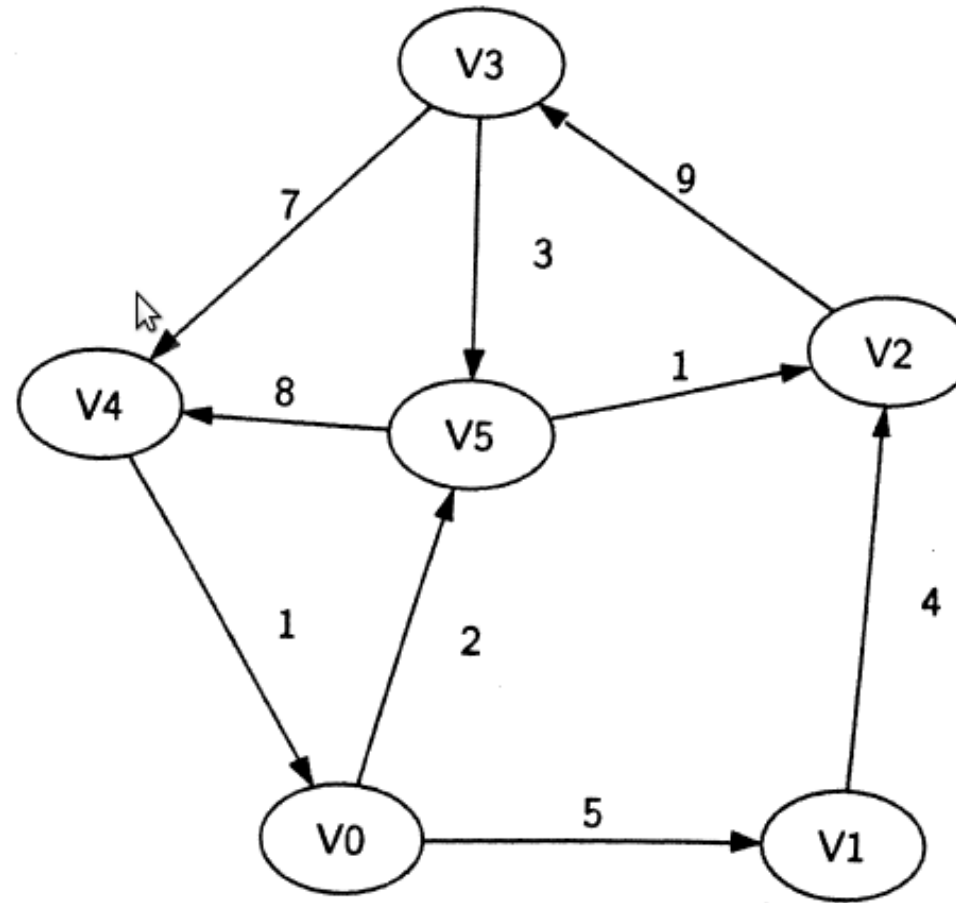
→ ağırlıklandırıldığında ise

$$e = (v, w, \text{ağırlık})$$

→ alt çizge  $e \in E$  ve  $v \in V$ 'lerden oluşan çizgedir



## örnek



**Figure 6.2:** A Simple Example of a Directed Graph

- ağırlıklı digraph'tır
- $V = \{V0, V1, \dots, V5\}$
- $E = \{(v0, v1, 5), (v1, v2, 4), \dots, (v5, v2, 1)\}$

# devam

path:

→ yol, girişlerle bağlanan düğümler serisi

$$w1, w2, w3, ..., wn \quad (wi, wi+1) \in E, i=1,...,n$$

→ ağırlıksız yol uzunluğu (AlıYU) = giriş sayısı =  $(n - 1)$

→ ağırlıklı yol uzunluğu (AsızYU) = Toplam ağırlık<sub>i</sub> \* giriş<sub>i</sub>

→ örnek:  $V3 \implies V1 : (V3, V4, V0, V1)$

→ burada AlıYU, AsızYU?

# devam

cycle:

- döngü, başlangıç ve bitimi aynı olan yol
- ör. (**V5**, V2, V3, **V5**)
- eğer çizge hiçbir döngü içermiyorsa “acyclic” graph denir
- yönlü acyclic graph “DAG” (directed acyclic graph) adlanır

## 6.3 temsil

çizge arayüzü

**Graph():** boş yeni çizge

**addVertex(Vert):** çizgeye düğüm ekle. Vert is instance of Vertex

**getVertex(vertKey):** çizgede vertKey'li düğümü bul

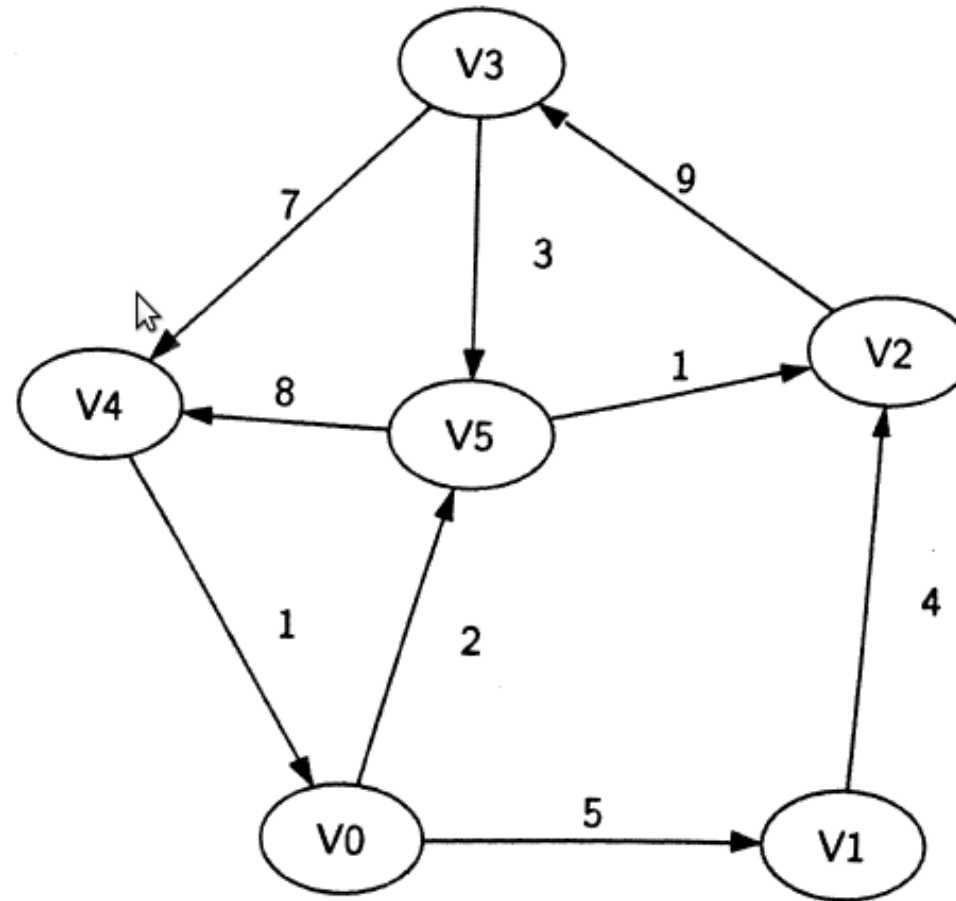
**getVertices():** çizgedeki düğümler listesini döndür.

→ çok sayıda temsil söz konusu

→ en iyi bilineni “komşuluk matrisi” (adjacency matrix)

## 6.3.1 komşuluk matrisi

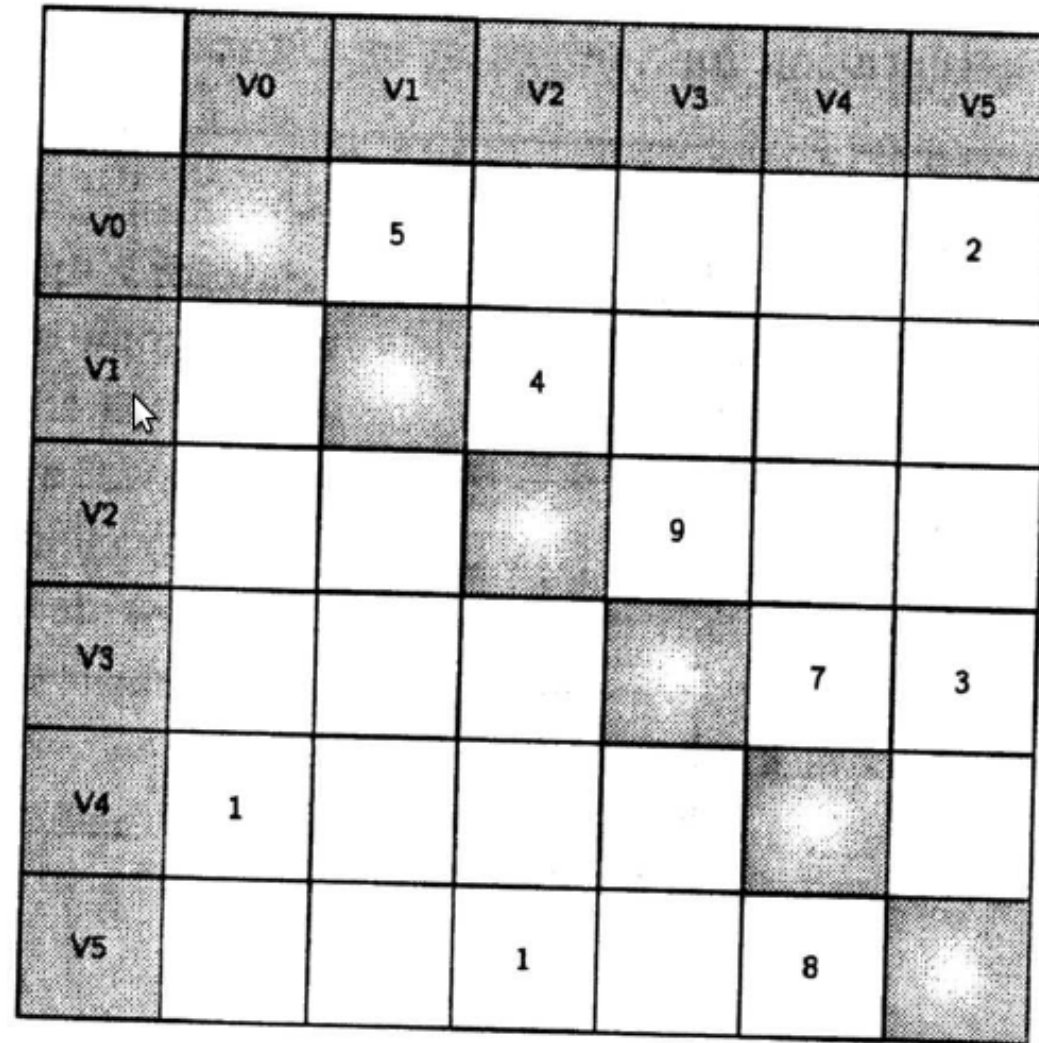
aşağıdaki resmin,



**Figure 6.2:** A Simple Example of a Directed Graph

örnek

komşuluk matrisi,



A 7x7 grid representing an adjacency matrix for a graph with 6 vertices (V0 to V5). The top row and left column are headers. The diagonal cells (V0-V0, V1-V1, V2-V2, V3-V3, V4-V4, V5-V5) are shaded gray. The matrix is symmetric, indicating an undirected graph. The values in the cells represent the weight of the edges between the vertices.

	V0	V1	V2	V3	V4	V5
V0		5				2
V1			4			
V2				9		
V3					7	3
V4	1					
V5			1		8	

Figure 6.3: An Adjacency Matrix Representation for a Graph

# komşuluk matrisi

- her bir düğüm arasındaki bağlantıyı gösterir
- köşegen boş
- kesişen yerlerdeki sayılar = ağırlıklar
- ör.  $V_4 \Rightarrow V_0$  kirişi için: satır= $V_4$ , sütun= $V_0 \Rightarrow 1$  (ağırlık)

# komşuluk matrisi

- avantajı: basittir
- küçük çizgeler kolaylıkla temsil edilebilir
- dezavantajı: matrisin çoğu boş, “sparse” (seyrek)
- seyrek veri için matris verimli olmaz
- komşuluk matrisi, giriş sayısı çoksa avantajlı/verimli olur
- tüm düğümlerin diğerleriyle bağlantısı pek rastlanılan bir durum değildir