# Graphs

CS 225 - Fall 2025

Mattox Beckman / Based on slides from Brad Solomon

# Objectives

Introduction

### Objectives

- · Define vocabulary needed to describe graphs
- · Implement a graph using an edge list

· Informal Early Feedback in the Discord

A graph G is a set V, E of vertices and edges

A graph G is a set V, E of vertices and edges

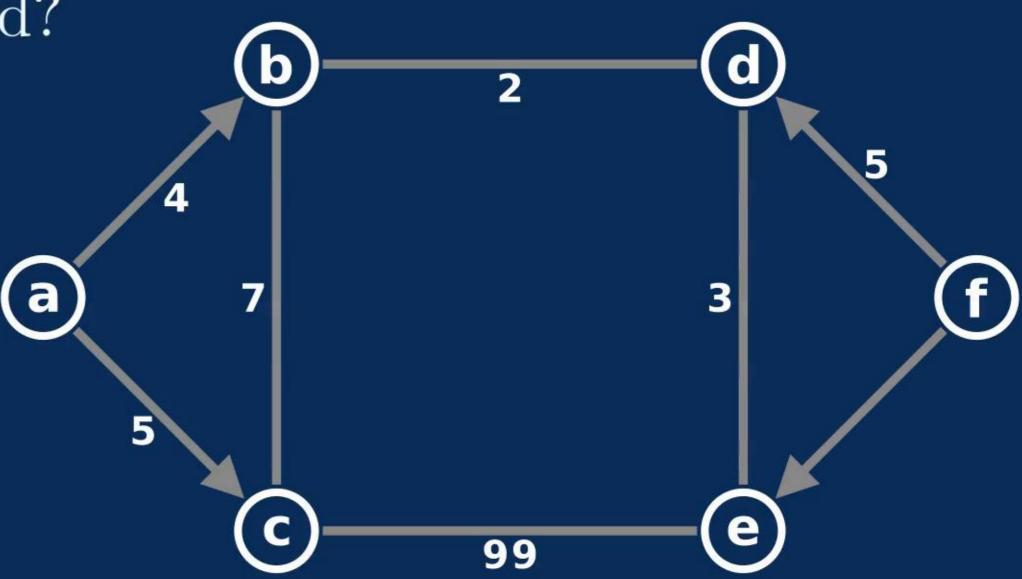
- · Vertices also known as nodes.
- · Can contain data; key-values; weights; etc.

### A graph G is a set V, E of vertices and edges

- · Vertices also known as nodes.
- · Can contain data; key-values; weights; etc.
- · Edges connect nodes; defined by their endpoints.
- · Edges can be directed or undirected, weighted or unweighted

Introduction

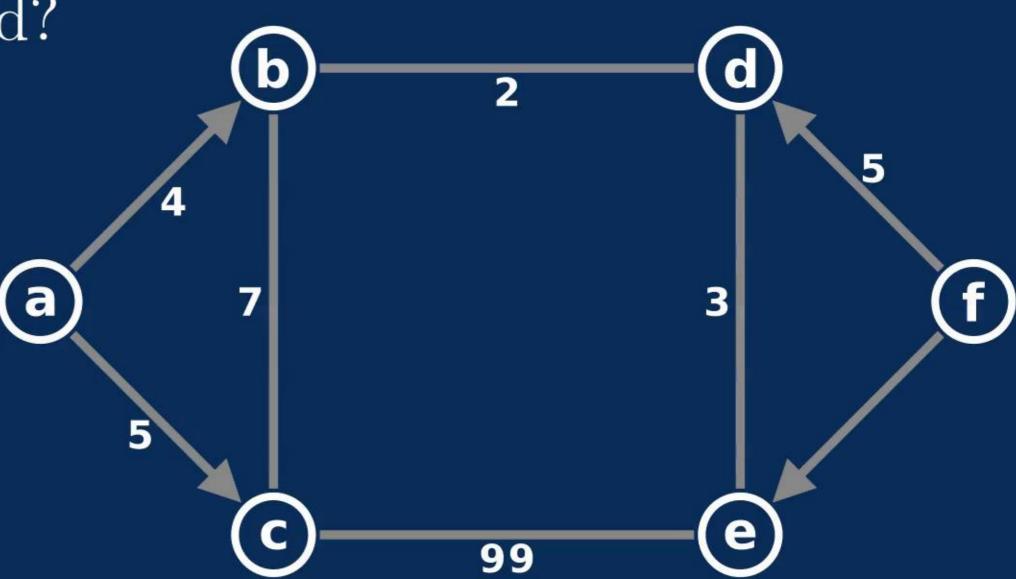
Which edges are weighted?



Introduction

Which edges are weighted?

· All but f-e.



Introduction

Which edges are weighted? b · All but f-e. Which edges are directed? 99

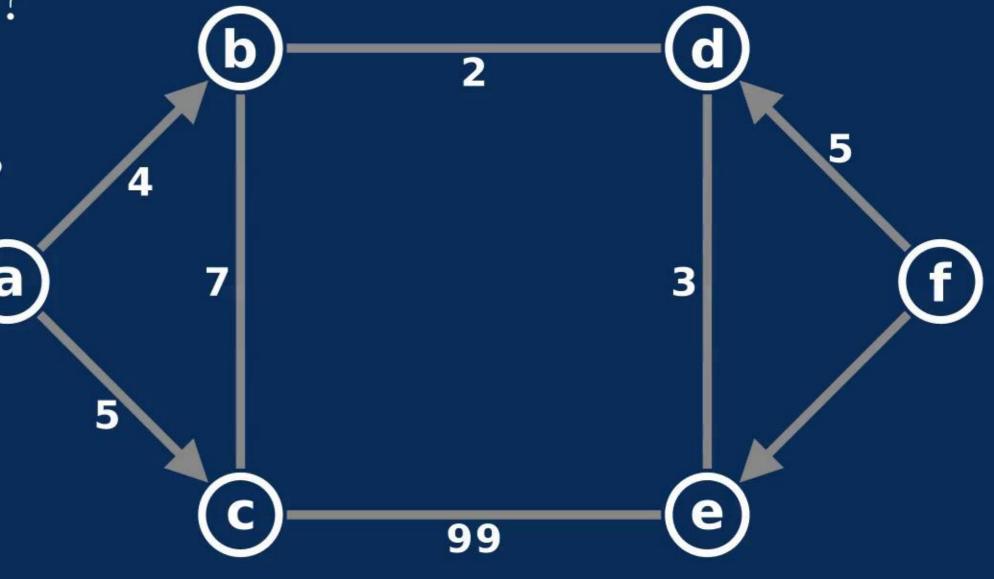
Introduction

Which edges are weighted?

· All but f-e.

Which edges are directed?

· a-b, a-c, f-d, f-e



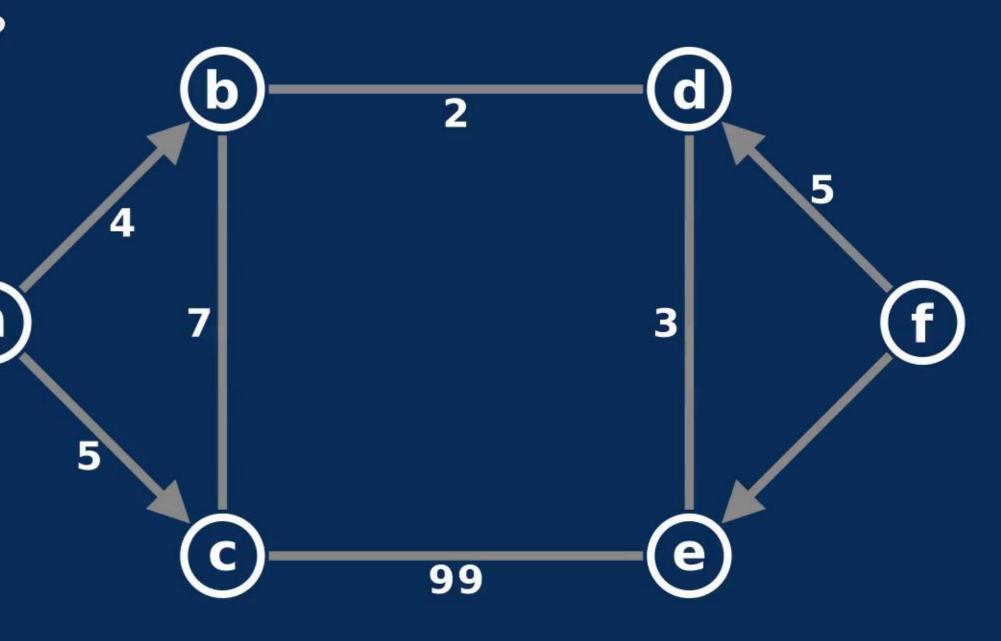
Introduction

Which edges are weighted?

· All but f-e.

Which edges are directed?

· a-b, a-c, f-d, f-e



## Uses of Graphs

Introduction

For these: what are the vertices and what are the edges?

- · Flight routes between cities
- · Legal Game moves
- · Friendship connections
- · Subway station map
- · Prerequisite maps for classes

Degree: number of incident edges

 $\cdot \deg(a) = 4, \deg(g) = 2$ 

Adjacent: connected by edge

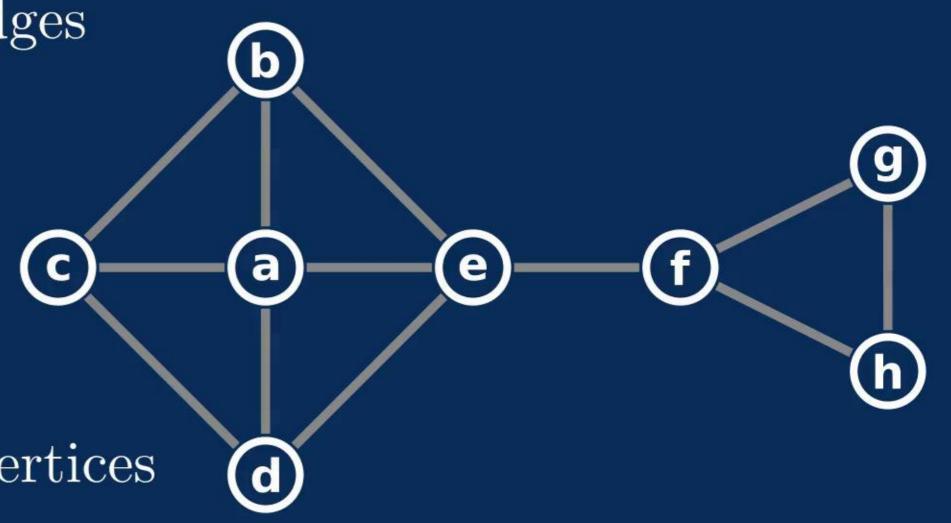
- · c is adjacent to a
- · c is not adjacent to f

Path: sequence of connected vertices

· e.g. c-a-e-f

Cycle: path that returns to start

· e.g. f-g-h



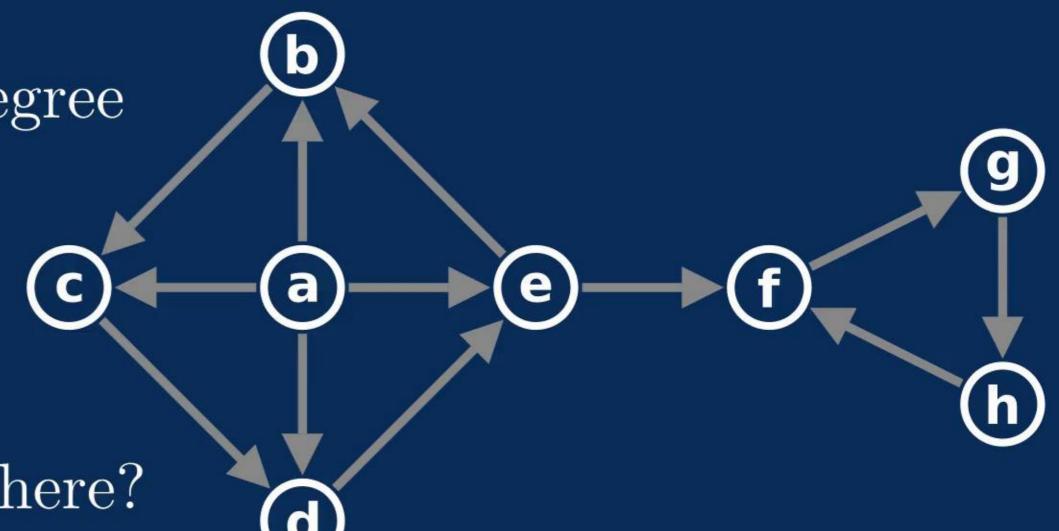
## Vocabulary

## Directed Graph:

- · In Degree and Out Degree
- · a has out-degree of 4
- · f has in-degree 2
- f has out-degree 1

Reachable: can I get there?

· g is reachable from a, but a is not reachable from g



## Simple Graphs

Introduction

Self Loops are possible

Multiple Edges are possible

We don't usually like those.

### Simple Graph:

- · No self loops!
- · No multiple edges!



A subgraph G' of G:

$$G' = (V', E')$$

$$\cdot V' \subset V$$

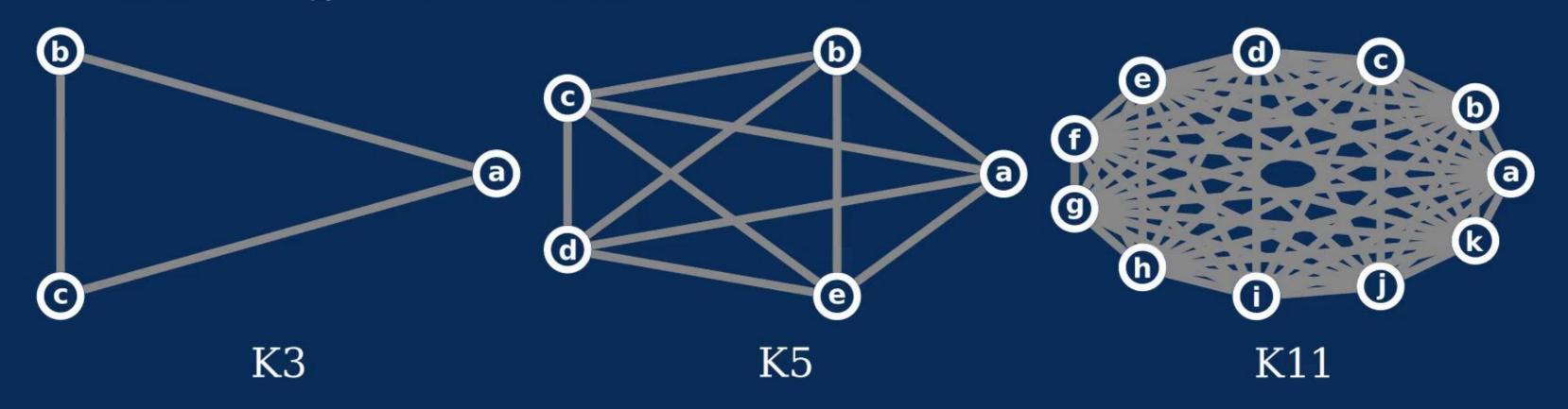
$$\cdot E' \subset E$$

if  $(v_1, v_2) \in E$ , then  $v_1 \in E$  and  $v_2 \in V'$ 

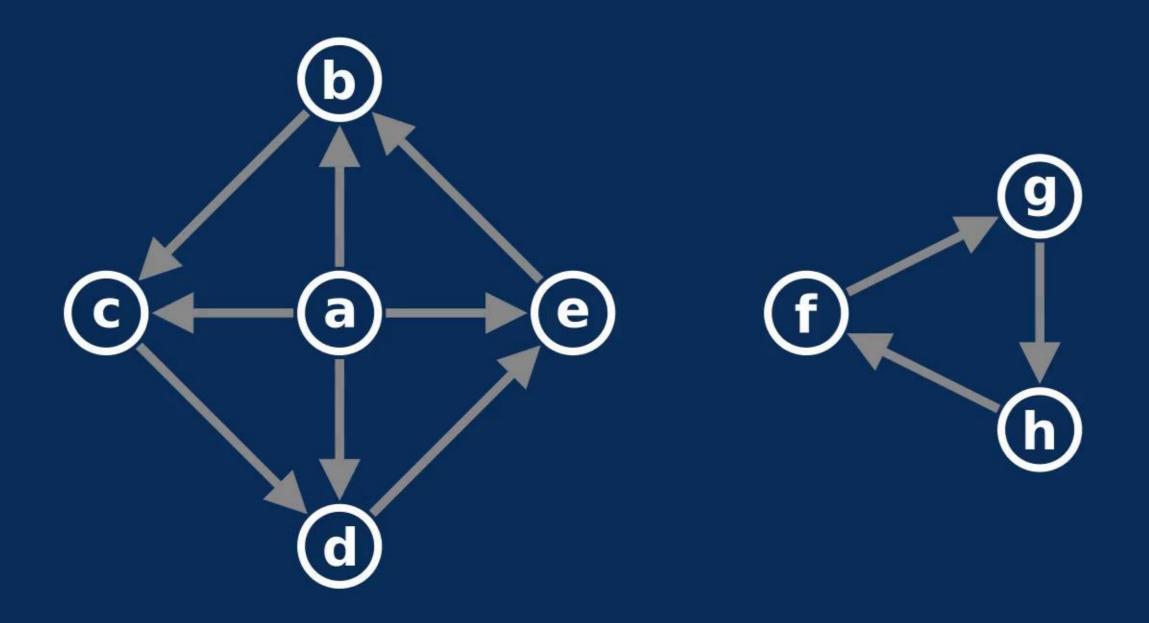
### Complete Graph:

$$\forall v_1, v_2 \in V.(v_1, v_2) \in E$$

Name  $K_n$  for n number of nodes.



A graph can be disconnected!

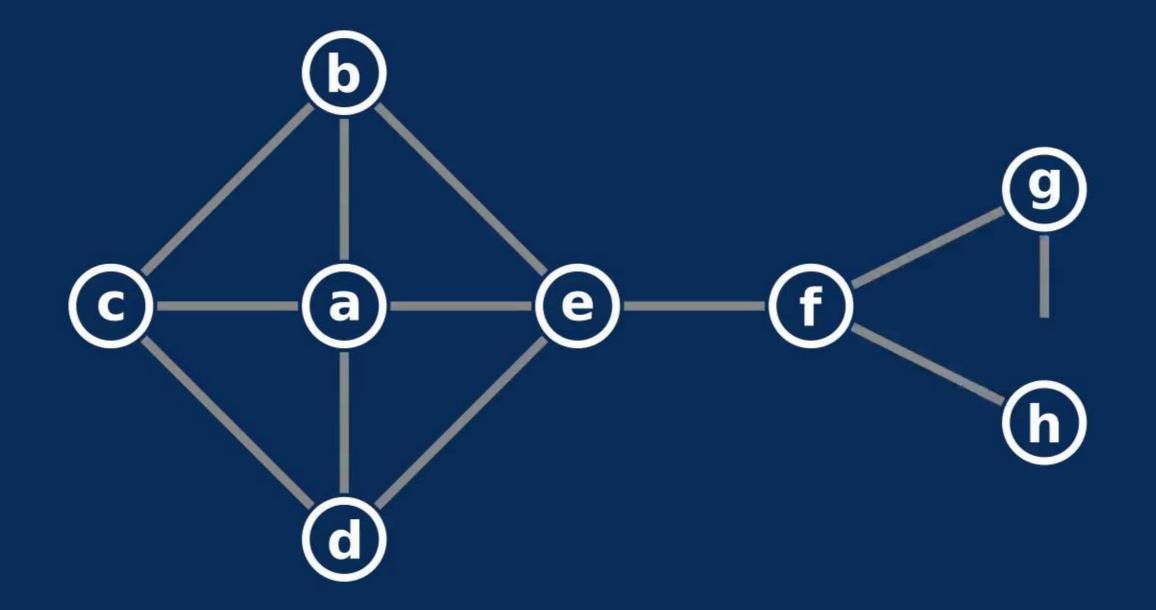


## **Connected Graphs**

Introduction

Every connected graph has a spanning tree.

There may be more than one!

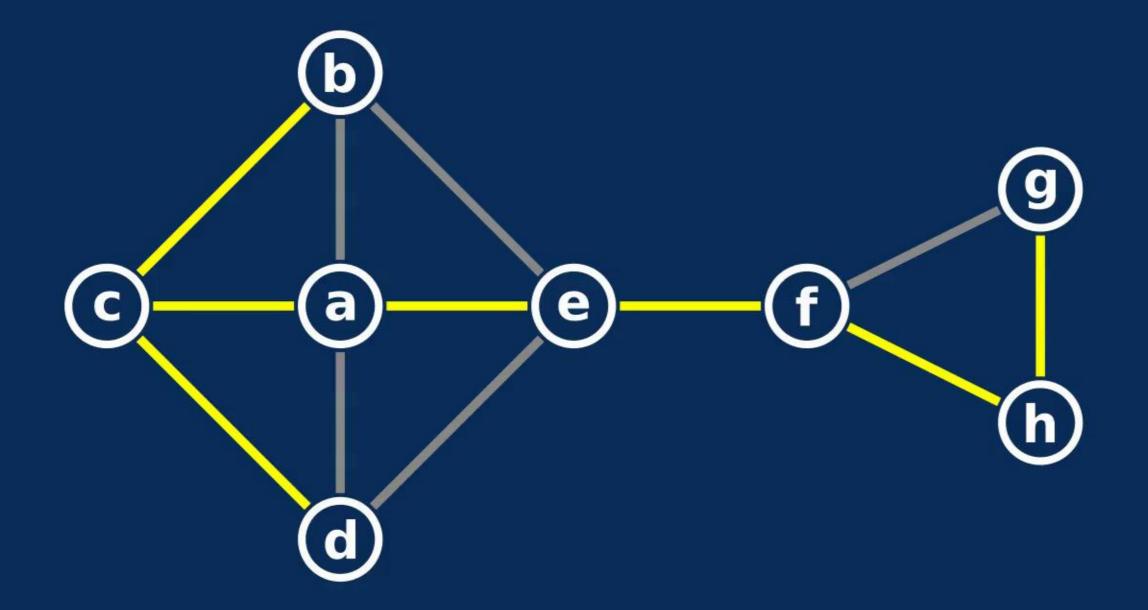


## **Connected Graphs**

Introduction

Every connected graph has a spanning tree.

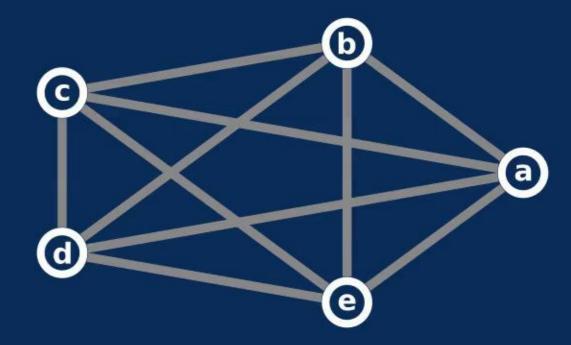
There may be more than one!



# Some Mathy Things

What do we need for the runtime of graphs?

- · Need |V| = n and also |E| = m
- So. How many edges can there be (min, max)?
- $\cdot$  min = 0; completely disconnected graph
- $\cdot \max = \mathcal{O}(n^2)$ ; complete graph



# Implementation

## **Graph ADT**

### Modification

- insertVertex(K key)
- · insertEdge(Vertex v1, Vertex v2, K Key)
- removeVertex(K key)
- removeEdge(Vertex v1, Vertex v2)

Need storage for vertices and keys. Maybe weights.

## **Graph ADT**

Implementations

### Query

- getEdges(Vertex v)
- · areAdjecent(Vertex v1, Vertex v2)
- · origin(Edge e)
- · destination(Edge e)

Three common ways to implement graphs

- · Edge List (vector of edges)
- · Adjacency List (vector of lists)
- · Adjacency Matrix (2d vector)

How do you pick?

Three common ways to implement graphs

- · Edge List (vector of edges)
- · Adjacency List (vector of lists)
- · Adjacency Matrix (2d vector)

How do you pick?

Data Representation Affects the Questions You Can Ask

## How To Do It

What needs to go in an edge?

### How To Do It

What needs to go in an edge?

· Source and Destination Vertices

What needs to go in an edge?

- · Source and Destination Vertices
- · Possibly a weight

## How To Do It

What needs to go in an edge?

- · Source and Destination Vertices
- · Possibly a weight
- · Possibly a key

### How To Do It

What needs to go in an edge?

- · Source and Destination Vertices
- · Possibly a weight
- · Possibly a key

What about direction?

What needs to go in an edge?

- · Source and Destination Vertices
- · Possibly a weight
- · Possibly a key

What about direction?

· Undirected graphs store edges twice.

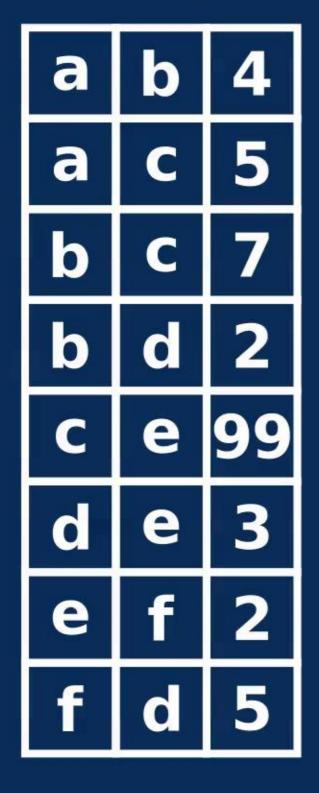
### How To Do It

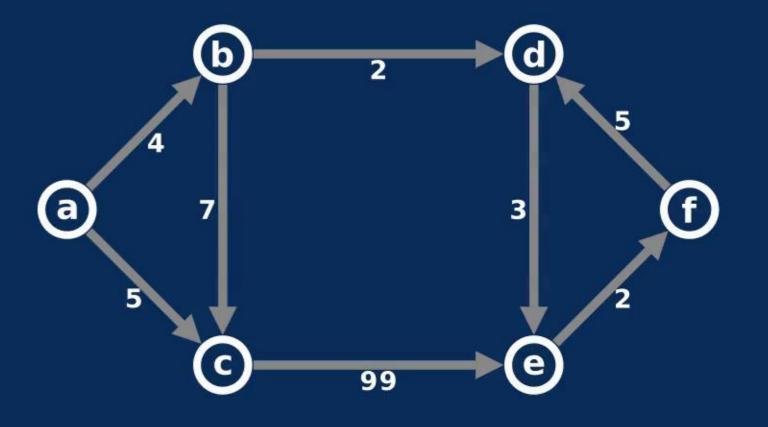
What needs to go in an edge?

- · Source and Destination Vertices
- · Possibly a weight
- · Possibly a key

What about direction?

· Undirected graphs store edges twice.







a b 4

b | c | 7

b | d | 2

c e 99

d e 3

e | f | 2

f d 5

How long do things take?

insertVertex(K key)

insertEdge(Vertex v1, Vertex v2, K Key)

removeVertex(K key)

removeEdge(Vertex v1, Vertex v2)

a b c d e f

How long do things take?

insertVertex(K key)

 $\cdot \mathcal{O}(1)$ 

insertEdge(Vertex v1, Vertex v2, K Key)

removeVertex(K key)

removeEdge(Vertex v1, Vertex v2)

a b c d e f

How long do things take?

insertVertex(K key)

 $\cdot \mathcal{O}(1)$ 

insertEdge(Vertex v1, Vertex v2, K Key)

 $\cdot \mathcal{O}(m)$ 

removeVertex(K key)

removeEdge(Vertex v1, Vertex v2)

a b c d e f

How long do things take?

insertVertex(K key)

 $\cdot \mathcal{O}(1)$ 

insertEdge(Vertex v1, Vertex v2, K Key)

 $\cdot \mathcal{O}(m)$ 

removeVertex(K key)

·  $\mathcal{O}(m)$  (need to remove edges)

removeEdge(Vertex v1, Vertex v2)

a b c d e f

How long do things take?

insertVertex(K key)

 $\cdot \mathcal{O}(1)$ 

insertEdge(Vertex v1, Vertex v2, K Key)

 $\cdot \mathcal{O}(m)$ 

removeVertex(K key)

•  $\mathcal{O}(m)$  (need to remove edges)

removeEdge(Vertex v1, Vertex v2)

 $\cdot \mathcal{O}(m)$  (need to remove edges)

a b c d e f

How long do things take?

getEdges(Vertex v)

areAdjacent(Vertex v1, Vertex v2)

origin(Edge e)

a b c d e f

origin(Edge e)

Edge List

How long do things take? getEdges(Vertex v)  $\cdot \mathcal{O}(m)$  - Check both start and end! areAdjacent(Vertex v1, Vertex v2)

origin(Edge e)

Edge List

How long do things take? getEdges(Vertex v)  $\cdot \mathcal{O}(m) \text{ - Check both start and end!}$  areAdjacent(Vertex v1, Vertex v2)  $\cdot \mathcal{O}(m)$ 

a b c d e f

How long do things take?

getEdges(Vertex v)

 $\cdot$   $\mathcal{O}(m)$  - Check both start and end! areAdjacent(Vertex v1, Vertex v2)

 $\cdot \mathcal{O}(m)$ 

origin(Edge e)

·  $\mathcal{O}(m)$  or  $\mathcal{O}(1)$ 

a b c d e f

#### Pros

- · Compact representation
- · Edge iteration is easy
- · Good for Minimum Spanning Trees

#### Cons

- · DFS and BFS are slow!
- · Vertex oriented things are slow

### Next Time

· Adjacency List and Adjacency Matrix