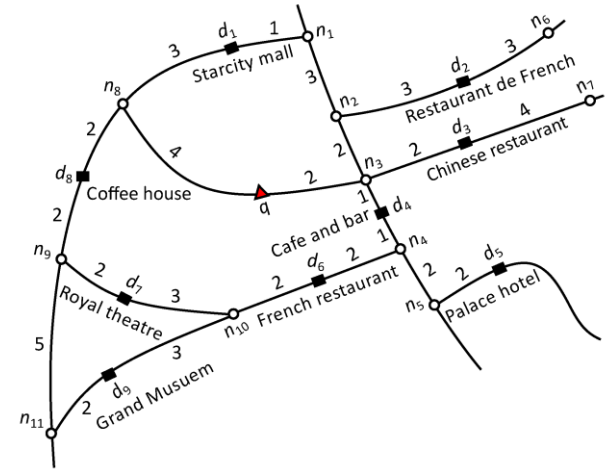


# Graph Representations

Alexander Lam

# Graphs

- How can we represent a graph?
  - There are two parts:
    - **Nodes**: can be represented as a list or a set.
    - **Edges**: can be represented as a list or a set.
      - For each edge, we would need a pair or a tuple expressing the two nodes.
  - Some graphs need more information on the edges:
    - **Weight** of an edge.
    - **Direction** of an edge.



# Graphs

- A graph  $G$  is denoted by:
  - $G = (V, E)$ 
    - $V$  is a set of vertices or nodes.
    - $E$  is a set of edges.

- Example

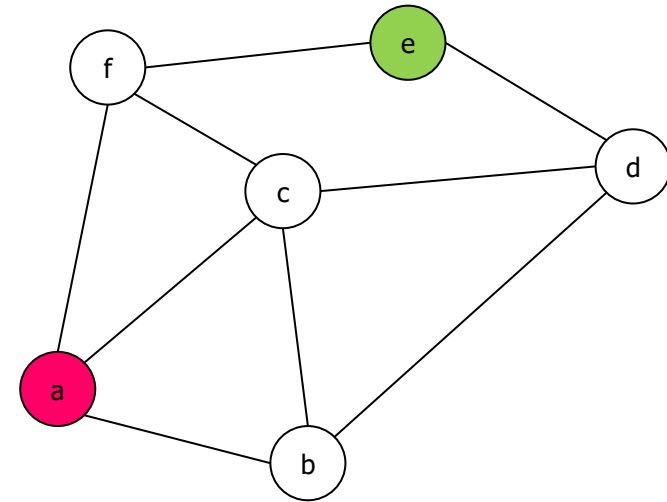
- Nodes

- $\{a, b, c, d, e, f\}$  (6 nodes)

- Edges

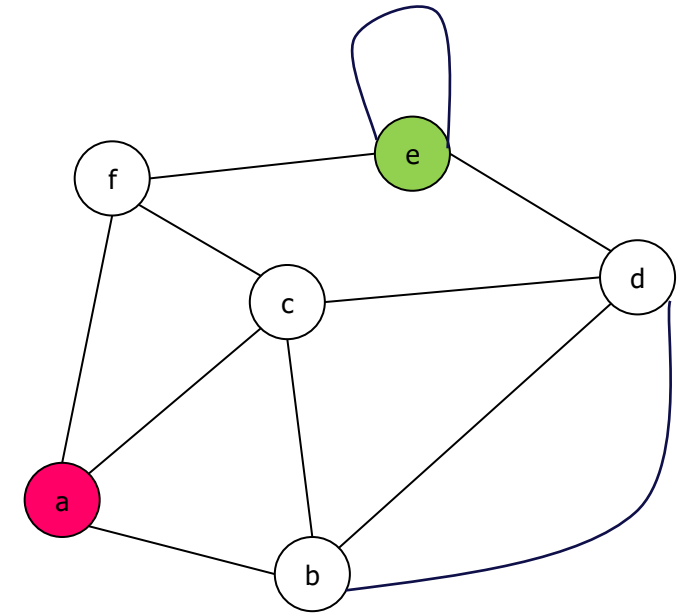
- $\{ab, ac, af, bc, bd, cd, cf, de, ef\}$  (9 edges)

- This graph is **undirected** (no direction on edges) and **unweighted** (no weights on edges).



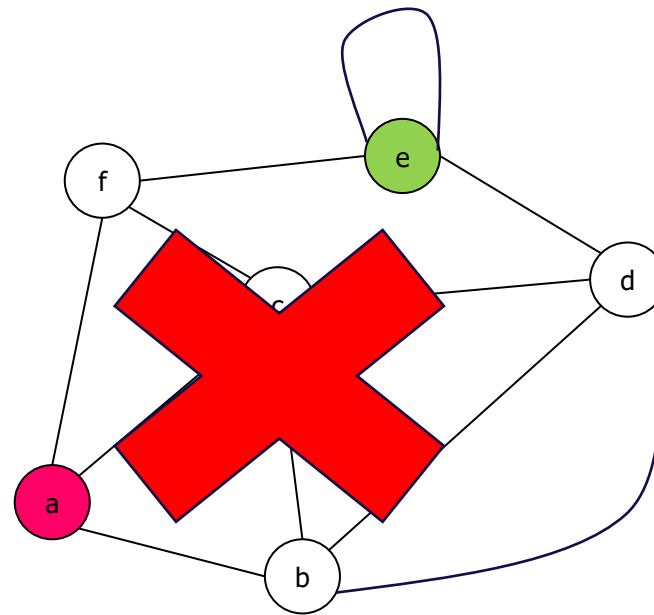
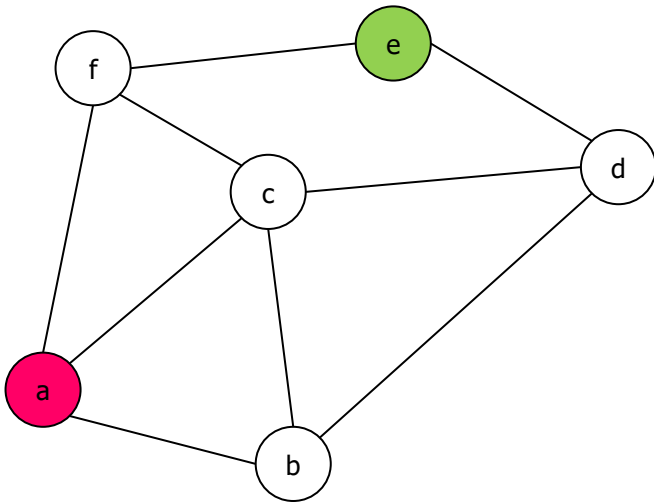
# Graphs

- By definition, an edge connects two vertices.
- Technically speaking, those two vertices can be the same.
  - An edge connecting a vertex with itself is called a **loop**
- We can also have multiple edges between the same two vertices.
- **Nodes:**
  - $\{a,b,c,d,e,f\}$  (6 nodes)
- **Edges:**
  - $\{ab, ac, af, bc, bd, \textcolor{blue}{bd}, cd, cf, de, \textcolor{blue}{ee}, ef\}$  (11 edges)



# Graphs

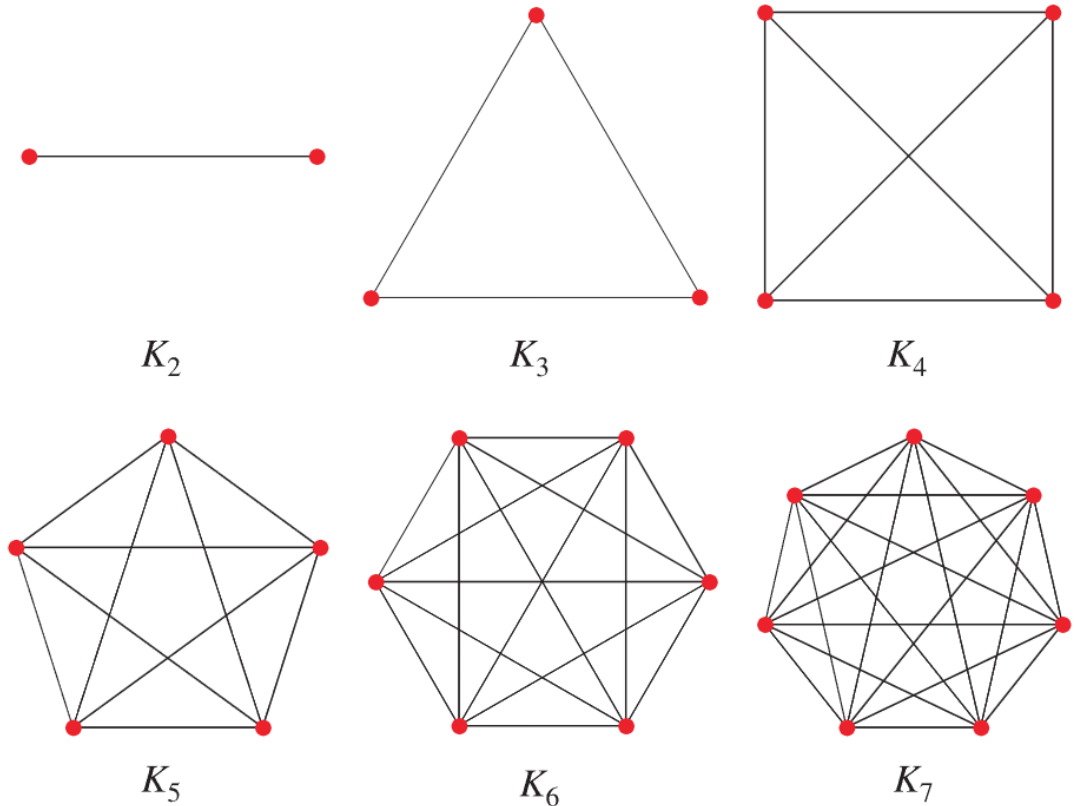
A **simple graph** is an unweighted, undirected graph with no loops and no multiple edges between the same pair of vertices



# Graphs

A **complete graph** is a graph where **each** pair of vertices is connected by an edge.

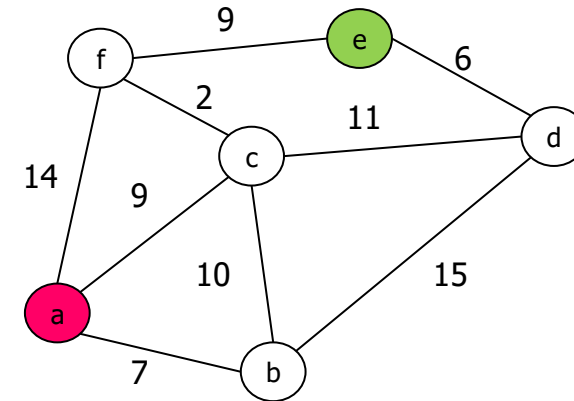
If a complete graph has  $v$  vertices, how many edges does it have?



# Graphs

Graphs can also be **weighted**

- **Nodes**
  - $\{a, b, c, d, e, f\}$  (6 nodes)
- **Edges**
  - $\{ab, ac, af, bc, bd, cd, cf, de, ef\}$  (9 edges)
- With **weights** on edges:
  - $\{ab=7, ac=9, af=14, bc=10, bd=15, cd=11, cf=2, de=6, ef=9\}$
- This graph is **undirected** (**no direction** on edges).



# Graphs

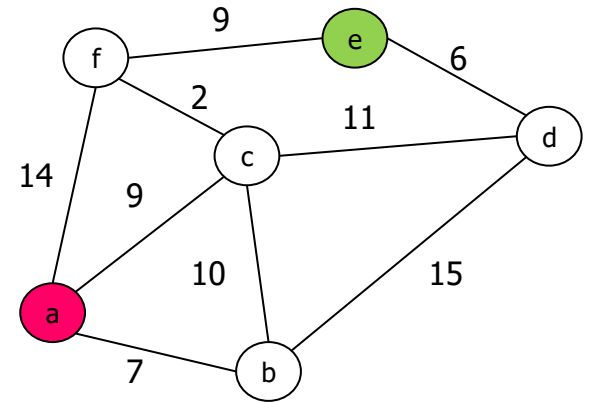
We need to build the **data model** for the **graph** before we can use the computer to process it.

- It is easy to represent the **nodes**, but perhaps harder with the **edges**.
- We may represent nodes and edges separately as two different groups.
- Let's try a naïve representation of the previous graph.



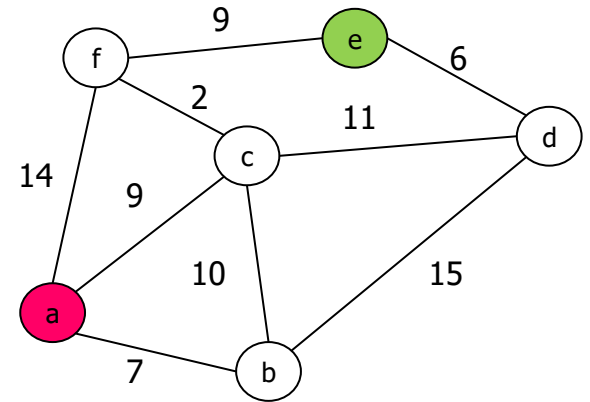
# Graphs

- Set of **nodes**:
  - $V = \{a, b, c, d, e, f\}$
- Set of **edges**:
  - $E = \{ab, ac, af, bc, bd, cd, cf, de, ef\}$
- Edges storing the weights:
  - $E = \{ab=7, ac=9, af=14, bc=10, bd=15, cd=11, cf=2, de=6, ef=9\}$
- Should we try to store nodes and edges **together** instead of **separately**?



# Graphs

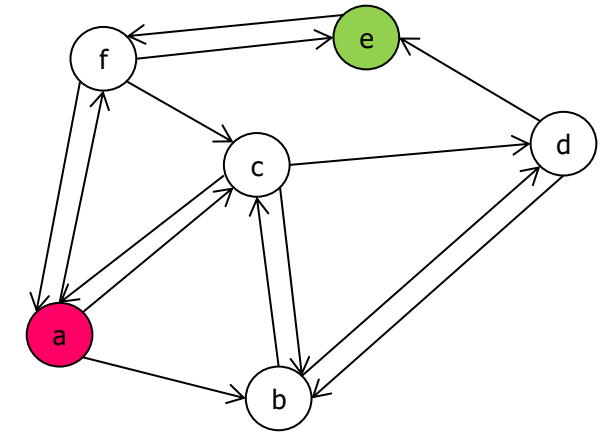
- Set of **nodes**:
  - $V = \{a, b, c, d, e, f\}$
- Set of **edges**:
  - $E = \{(a, b), (a, c), (a, f), (b, c), (b, d), (c, d), (c, f), (d, e), (e, f)\}$
- Edges storing the weights:
  - $E = \{(a, b, 7), (a, c, 9), (a, f, 14), (b, c, 10), (b, d, 15), (c, d, 11), (c, f, 2), (d, e, 6), (e, f, 9)\}$
- What if the edges have directions?



# Graphs

Graphs can be **directed**

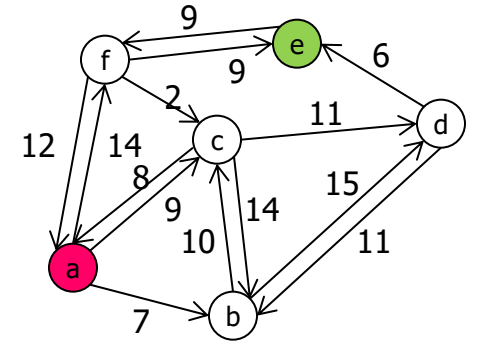
- Set of **nodes**:
  - $V = \{a, b, c, d, e, f\}$
- Set of **edges**:
  - $E = \{(a, b), (a, c), (a, f), (b, c), (b, d), (c, a), (c, b), (c, d), \textcolor{red}{(c, f)}, \textcolor{blue}{(d, b)}, (d, e), (e, f), \textcolor{blue}{(f, a)}, \textcolor{blue}{(f, c)}, \textcolor{blue}{(f, e)}\}$
- Note that an **edge** in an **undirected graph** is equivalent to a **pair of edges** in a **directed graph**.



# Graphs

Graphs can be **directed** and **weighted**

- Set of **nodes**:
  - $V = \{a, b, c, d, e, f\}$
- Set of **edges**:
  - $E = \{(a, b, 7), (a, c, 9), (a, f, 14), (b, c, 10), (b, d, 15), (c, a, 8), (c, b, 14), (c, d, 11), (d, b, 11), (d, e, 6), (e, f, 9), (f, a, 12), (f, c, 2), (f, e, 9)\}$



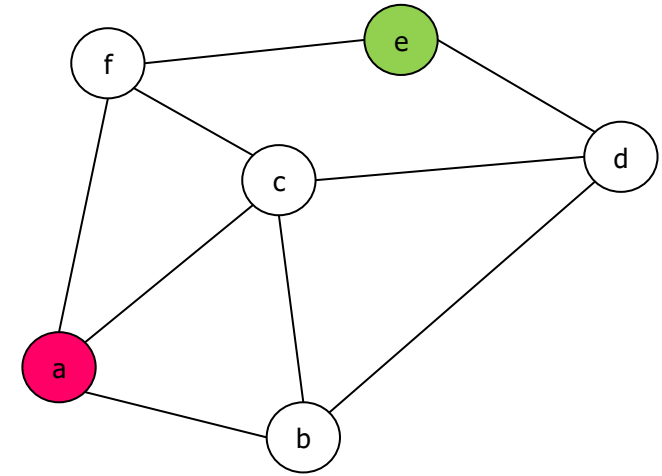
# Graphs

It is more natural to represent **edges as linked to nodes**. There are two common representations.

- Adjacency list
- Adjacency matrix

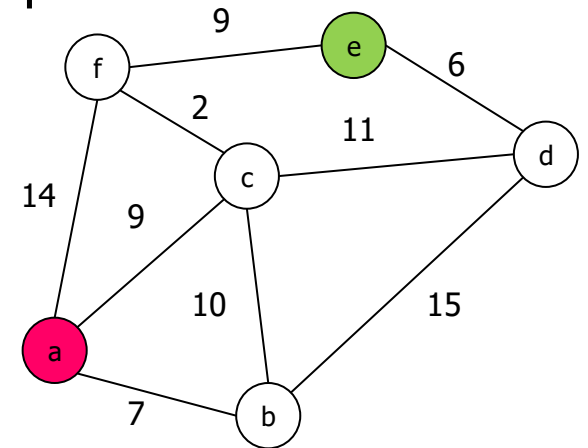
# Adjacency Lists

- For each node, put together the edges for each node.
- An **adjacency list** is a **list for each node**, showing the neighbors of that node, i.e. edges.
- Example
  - **Nodes**, a set
    - $\{a, b, c, d, e, f\}$
  - **Edges** in 6 adjacency lists:  $D(\text{node})$ , each is a set
    - $D(a) = \{b, c, f\}$
    - $D(b) = \{a, c, d\}$
    - $D(c) = \{a, b, d, f\}$
    - $D(d) = \{b, c, e\}$
    - $D(e) = \{d, f\}$
    - $D(f) = \{a, c, e\}$



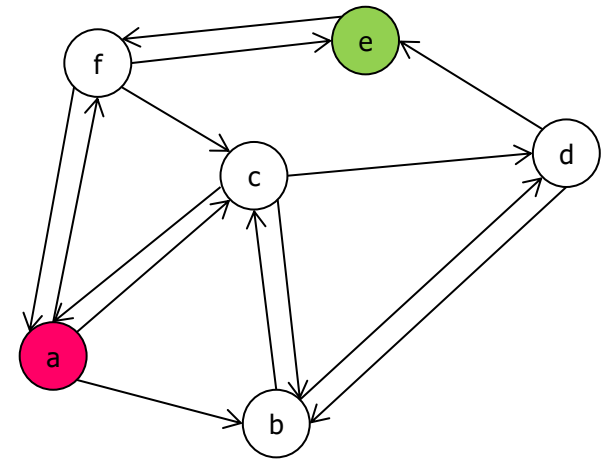
# Adjacency Lists

- For edges with **weights**, an adjacency list is a list for each node, showing the **edges and the weights**.
- Example
  - **Nodes**, a set
    - $\{a,b,c,d,e,f\}$
  - **Edges** in adjacency lists:  $D(\text{node})$ , each being a set of tuples
    - $D(a) = \{(b,7),(c,9),(f,14)\}$
    - $D(b) = \{(a,7),(c,10),(d,15)\}$
    - $D(c) = \{(a,9),(b,10),(d,11),(f,2)\}$
    - $D(d) = \{(b,15),(c,11),(e,6)\}$
    - $D(e) = \{(d,6),(f,9)\}$
    - $D(f) = \{(a,14),(c,2),(e,9)\}$



# Adjacency Lists

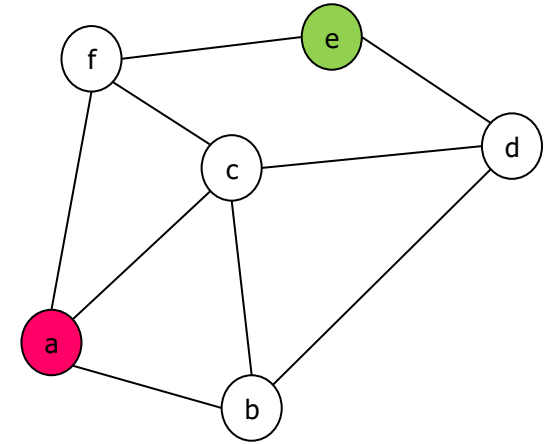
- For a **directed graph**, an adjacency list is a list for each node, showing the **next reachable node** and the **weights** (if any).
- Example
  - **Nodes**, a set
    - $\{a,b,c,d,e,f\}$
  - **Edges** in adjacency lists:  $D(\text{node})$ , each being a set
    - $D(a) = \{b,c,f\}$
    - $D(b) = \{c,d\}$
    - $D(c) = \{a,b,d\}$
    - $D(d) = \{b,e\}$
    - $D(e) = \{f\}$
    - $D(f) = \{a,c,e\}$





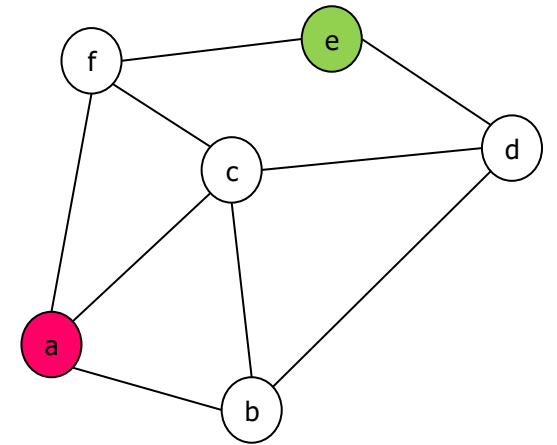
# Adjacency Lists

- Modeling in Python
  - Nodes can be represented as a list.
  - Edges can be represented as a list of lists.
- Example
  - Nodes as a list:
    - `N = ["a", "b", "c", "d", "e", "f"]`
  - Edges as a list of lists:
    - `D = [ ["b", "c", "f"],  
         ["a", "c", "d"],  
         ["a", "b", "d", "f"],  
         ["b", "c", "e"],  
         ["d", "f"],  
         ["a", "c", "e"] ]`



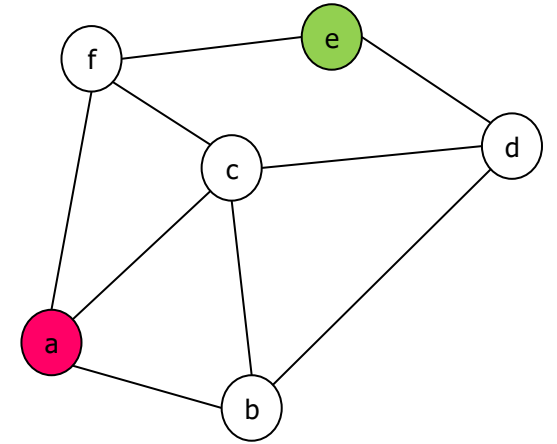
# Adjacency Lists

- Modeling in Python
  - Nodes can be represented as a **list**.
  - Edges can be represented as a **list of sets**.
- Example
  - Nodes as a **list**:
    - $N = ["a", "b", "c", "d", "e", "f"]$
  - Edges as a **list of sets**:
    - $D = [ \{ "b", "c", "f" \},$   
 $\{ "a", "c", "d" \},$   
 $\{ "a", "b", "d", "f" \},$   
 $\{ "b", "c", "e" \},$   
 $\{ "d", "f" \},$   
 $\{ "a", "c", "e" \} ]$



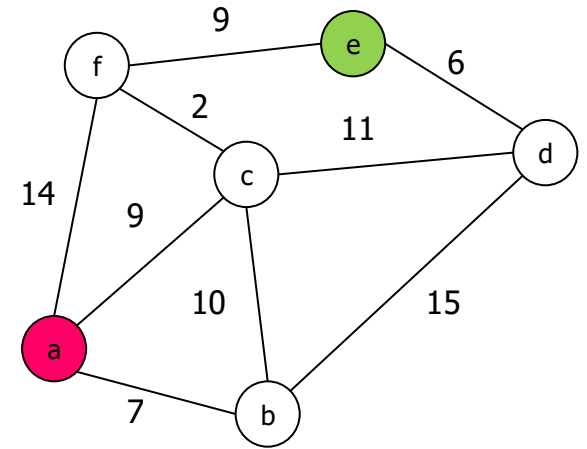
# Adjacency Lists

- Modeling in Python
  - **Nodes** can be represented as **dictionary keys**.
  - **Edges** can be represented as **dictionary values in list/set**.
- Example
  - **Node** and **edges** as a **dictionary**:
    - $D = \{ \text{"a": } \{ \text{"b"}, \text{"c"}, \text{"f"} \},$   
           $\text{"b": } \{ \text{"a"}, \text{"c"}, \text{"d"} \},$   
           $\text{"c": } \{ \text{"a"}, \text{"b"}, \text{"d"}, \text{"f"} \},$   
           $\text{"d": } \{ \text{"b"}, \text{"c"}, \text{"e"} \},$   
           $\text{"e": } \{ \text{"d"}, \text{"f"} \},$   
           $\text{"f": } \{ \text{"a"}, \text{"c"}, \text{"e"} \} \}$



# Adjacency Lists

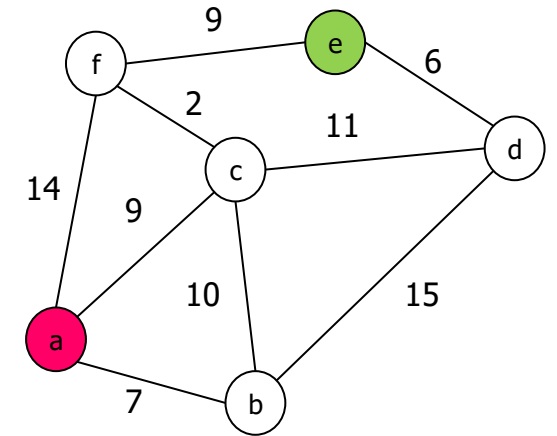
- Modeling in Python
  - **Nodes** can be represented as **dictionary keys**.
  - **Edges** can be represented as **dictionary values in list/set**.
- Example
  - **Node** and **edges** as a **dictionary of tuples**:
    - $D = \{ \text{"a": } \{(\text{"b"}, 7), (\text{"c"}, 9), (\text{"f"}, 14)\},$   
     $\text{"b": } \{(\text{"a"}, 7), (\text{"c"}, 10), (\text{"d"}, 15)\},$   
     $\text{"c": } \{(\text{"a"}, 9), (\text{"b"}, 10), (\text{"d"}, 11), (\text{"f"}, 2)\},$   
     $\text{"d": } \{(\text{"b"}, 15), (\text{"c"}, 11), (\text{"e"}, 6)\},$   
     $\text{"e": } \{(\text{"d"}, 6), (\text{"f"}, 9)\},$   
     $\text{"f": } \{(\text{"a"}, 14), (\text{"c"}, 2), (\text{"e"}, 9)\} \}$



# Adjacency Lists

## Other combinations of Python data models

- Modeling in Python
  - **Nodes** can be represented as a **list**.
  - **Edges** with weights can be represented as a **list of lists of tuples**.
- Example
  - **Nodes** as a **list**:
    - `N = ["a", "b", "c", "d", "e", "f"]`
  - **Edges** as a **list of lists of tuples**:
    - `D = [ [ ("b",7), ("c",9), ("f",14) ],  
[ ("a",7), ("c",10), ("d",15) ],  
[ ("a",9), ("b",10), ("d",11), ("f",2) ],  
[ ("b",15), ("c",11), ("e",6) ],  
[ ("d",6), ("f",9) ],  
[ ("a",14), ("c",2), ("e",9) ] ]`

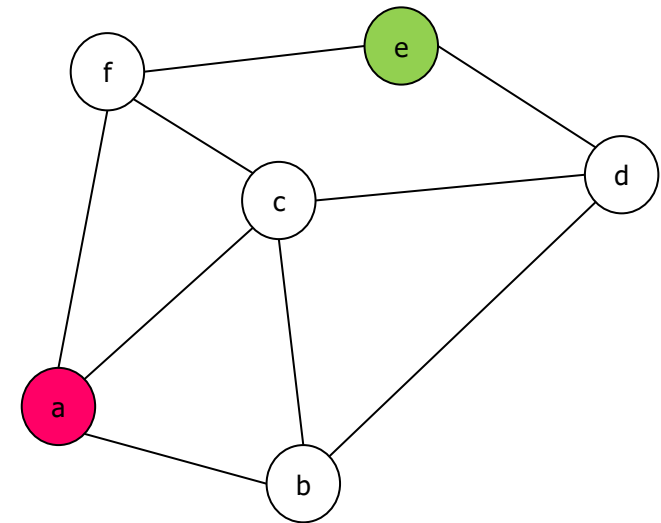


# Adjacency Matrices

- For **each pair of nodes**, maintain a **matrix** to show the neighborhood between the pair.
  - An **adjacency matrix** has each node in a row (source) and in a column (destination).
  - **1 / True** means an edge and **0 / False** means no edge.
- Example
  - Matrix  $D[6,6]$  for a graph with 6 nodes

•  $D =$

	<b>a</b>	<b>b</b>	<b>c</b>	<b>d</b>	<b>e</b>	<b>f</b>
<b>a</b>	0	1	1	0	0	1
<b>b</b>	1	0	1	1	0	0
<b>c</b>	1	1	0	1	0	1
<b>d</b>	0	1	1	0	1	0
<b>e</b>	0	0	0	1	0	1
<b>f</b>	1	0	1	0	1	0

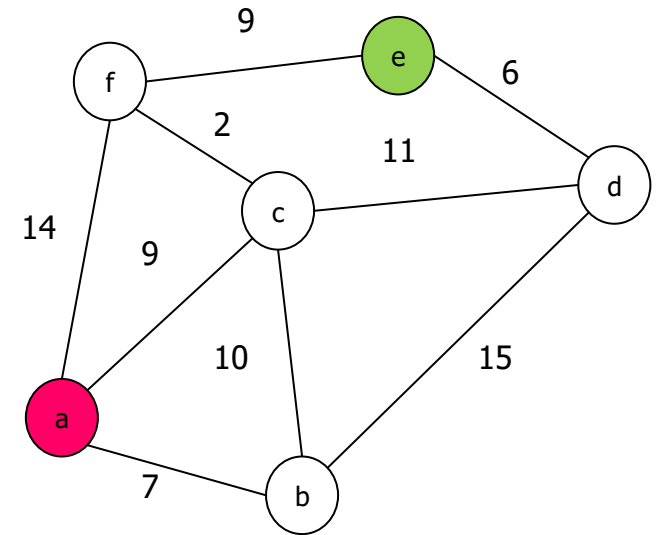


# Adjacency Matrices

- An **adjacency matrix** for a **weighted** graph has matrix elements showing neighborhood and **weight**.
- This is often called a **distance matrix** when the weight is the distance.
- Example
  - Matrix  $D[6,6]$  for a graph with 6 nodes

•  $D =$

	<b>a</b>	<b>b</b>	<b>c</b>	<b>d</b>	<b>e</b>	<b>f</b>
<b>a</b>	0	7	9	$\infty$	$\infty$	14
<b>b</b>	7	0	10	15	$\infty$	$\infty$
<b>c</b>	9	10	0	11	$\infty$	2
<b>d</b>	$\infty$	15	11	0	6	$\infty$
<b>e</b>	$\infty$	$\infty$	$\infty$	6	0	9
<b>f</b>	14	$\infty$	2	$\infty$	9	0



# Adjacency Matrices

- **Note:** The adjacency matrix for an undirected graph is **symmetric**
  - $D_{ij} = D_{ji}$

	a	b	c	d	e	f
a	0	1	1	0	0	1
b	1	0	1	1	0	0
c	1	1	0	1	0	1
d	0	1	1	0	1	0
e	0	0	0	1	0	1
f	1	0	1	0	1	0

	a	b	c	d	e	f
a	0	7	9	$\infty$	$\infty$	14
b	7	0	10	15	$\infty$	$\infty$
c	9	10	0	11	$\infty$	2
d	$\infty$	15	11	0	6	$\infty$
e	$\infty$	$\infty$	$\infty$	6	0	9
f	14	$\infty$	2	$\infty$	9	0

- Edge from a to b  $\iff$  Edge from b to a

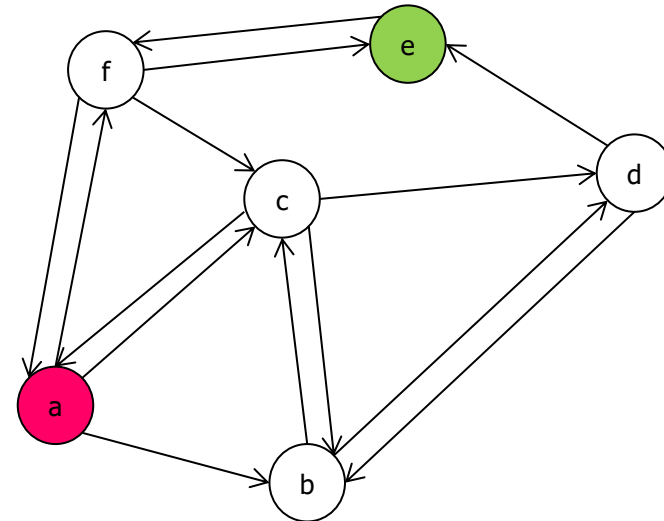


# Adjacency Matrices

- Example
  - Matrix  $D[6,6]$  for a **directed**, **unweighted** graph with 6 nodes

•  $D =$

	<b>a</b>	<b>b</b>	<b>c</b>	<b>d</b>	<b>e</b>	<b>f</b>
<b>a</b>	0	1	1	0	0	1
<b>b</b>	<b>0</b>	0	1	1	0	0
<b>c</b>	1	1	0	1	0	<b>0</b>
<b>d</b>	0	1	<b>0</b>	0	1	0
<b>e</b>	0	0	0	<b>0</b>	0	1
<b>f</b>	1	0	1	0	1	0

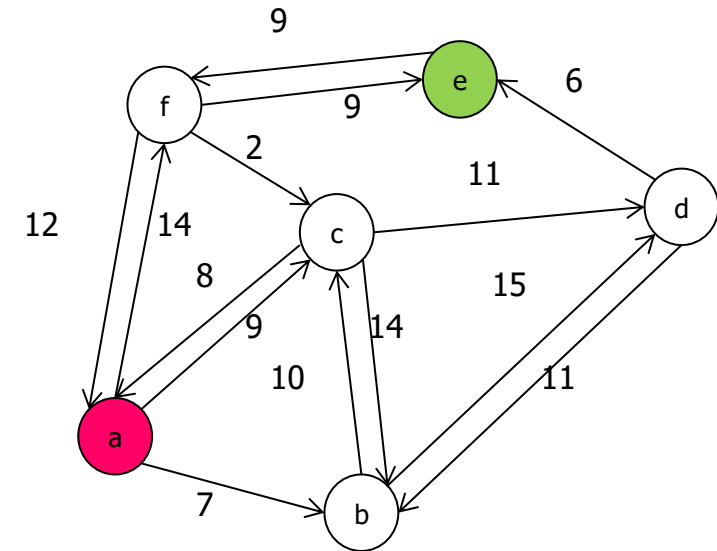


# Adjacency Matrices

- Example
  - Matrix  $D[6,6]$  for a **directed, weighted** graph with 6 nodes

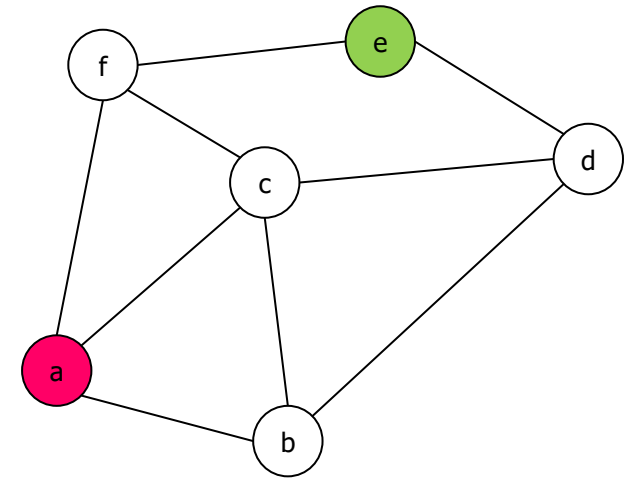
•  $D =$

	a	b	c	d	e	f
a	0	7	9	$\infty$	$\infty$	14
b	$\infty$	0	10	15	$\infty$	$\infty$
c	8	14	0	11	$\infty$	$\infty$
d	$\infty$	11	$\infty$	0	6	$\infty$
e	$\infty$	$\infty$	$\infty$	$\infty$	0	9
f	12	$\infty$	2	$\infty$	9	0



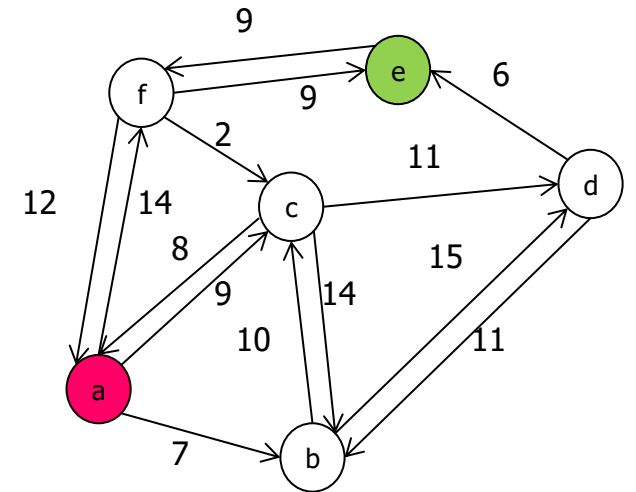
# Adjacency Matrices

- Modeling in Python
  - The 2-D matrix is normally represented as a list of lists as a logical 2-D array.
- Example
  - Nodes as a list:
    - $N = ['a', 'b', 'c', 'd', 'e', 'f']$
  - Matrix as a list of lists:
    - $D = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$

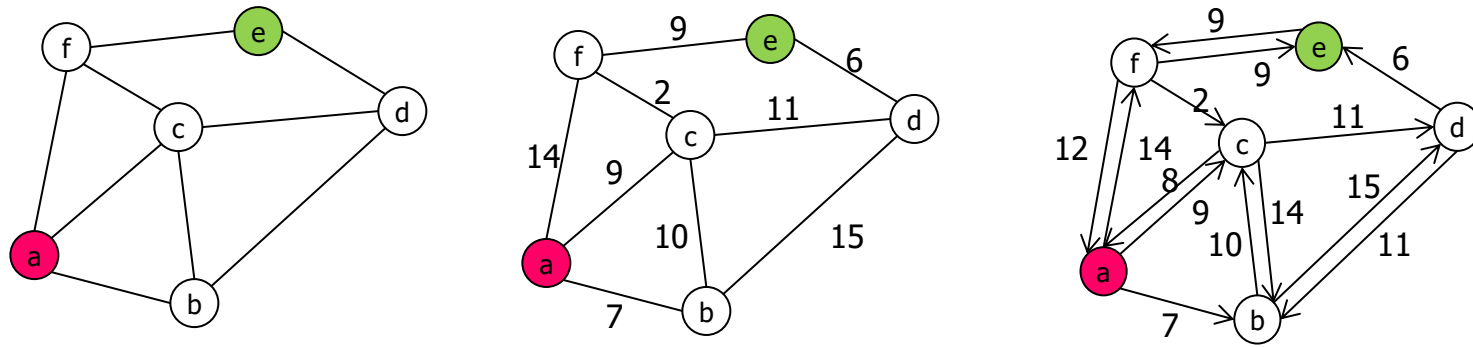


# Adjacency Matrices

- Modeling in Python
  - The 2-D matrix can also be represented as a dictionary of dictionaries.
- Example
  - Nodes as a list: **Not really needed** with dictionary D
    - $N = ["a", "b", "c", "d", "e", "f"]$
  - Matrix as a dictionary of dictionary.
    - $D = \{ \text{"a": } \{ \text{"a":0, "b":7, "c":9, "d":inf, "e":inf, "f":14} \},$   
     $\text{"b": } \{ \text{"a":inf, "b":0, "c":10, "d":15, "e":inf, "f":inf} \},$   
     $\text{"c": } \{ \text{"a":8, "b":14, "c":0, "d":11, "e":inf, "f":inf} \},$   
     $\text{"d": } \{ \text{"a":inf, "b":11, "c":inf, "d":0, "e":6, "f":inf} \},$   
     $\text{"e": } \{ \text{"a":inf, "b":inf, "c":inf, "d":inf, "e":0, "f":9} \},$   
     $\text{"f": } \{ \text{"a":12, "b":inf, "c":2, "d":inf, "e":9, "f":0} \} \}$
    - Here, `inf = float("infinity")` in Python, meaning infinity ( $\infty$ ).



# Shortest Paths

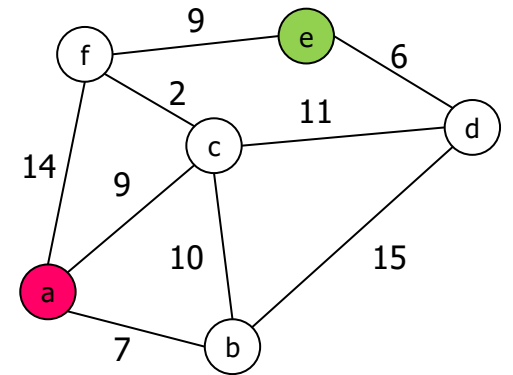


- Finding the **shortest path** from one node to another.
  - The **starting** node is often called **source** node in graph theory.
  - The **target** node is often called **destination** node.
  - The graphs may be **weighted** or **unweighted**
  - The graphs may be **directed** or **undirected**.
- There are other applications on a graph, e.g. **breadth first search**, **depth first search**, **topological sort**.



# Shortest Path

- Can you find the **shortest path** from **a** to **e**?
  - Algorithm 1: Layman approach to find **all possible paths** first.
    - $a \rightarrow f \rightarrow e$  ■ 23
    - $a \rightarrow c \rightarrow f \rightarrow e$  ■ 20
    - $a \rightarrow c \rightarrow d \rightarrow e$  ■ 26
    - $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$  ■ 34
    - $a \rightarrow b \rightarrow d \rightarrow e$  ■ 28
    - $a \rightarrow c \rightarrow b \rightarrow d \rightarrow e$  ■ 40
    - Any more?
    - $a \rightarrow b \rightarrow c \rightarrow f \rightarrow e$  ■ 28
    - $a \rightarrow b \rightarrow d \rightarrow c \rightarrow f \rightarrow e$  ■ 44
    - Yet more?
    - $a \rightarrow f \rightarrow c \rightarrow d \rightarrow e$  ■ 33
    - $a \rightarrow f \rightarrow c \rightarrow b \rightarrow d \rightarrow e$  ■ 47
  - **Very slow** for larger graphs!



# Fun with Adjacency Matrices

- Consider the adjacency matrix  $D$  for an **unweighted** graph.
  - The graph may be directed or undirected.

	<b>a</b>	<b>b</b>	<b>c</b>	<b>d</b>	<b>e</b>	<b>f</b>
<b>a</b>	0	1	1	0	0	1
<b>b</b>	1	0	1	1	0	0
<b>c</b>	1	1	0	1	0	1
<b>d</b>	0	1	1	0	1	0
<b>e</b>	0	0	0	1	0	1
<b>f</b>	1	0	1	0	1	0

- If there is a 1 in cell  $D_{ij}$ , then there is an edge from  $i$  to  $j$ .

# Fun with Adjacency Matrices

- Consider the matrix multiplication  $A = D \times D$

	a	b	c	d	e	f
a			?			
b						
c						
d						
e						
f						

=

	a	b	c	d	e	f
a	0	1	1	0	0	1
b	1	0	1	1	0	0
c	1	1	0	1	0	1
d	0	1	1	0	1	0
e	0	0	0	1	0	1
f	1	0	1	0	1	0

×

	a	b	c	d	e	f
a	0	1	1	0	0	1
b	1	0	1	1	0	0
c	1	1	0	1	0	1
d	0	1	1	0	1	0
e	0	0	0	1	0	1
f	1	0	1	0	1	0

- The result in cell  $A_{ac}$  is the dot-product of row a with column c.
- Consider the element-wise product vector (before they are summed).
  - 1 in the  $i$ th element means that there is an edge from a to i **and** there is an edge from i to c.



# Fun with Adjacency Matrices

- Consider the matrix multiplication  $A = D \times D$

	a	b	c	d	e	f
a			2			
b						
c						
d						
e						
f						

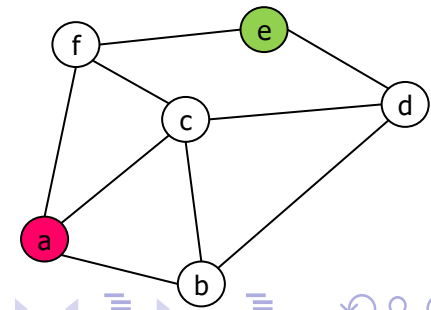
=

	a	b	c	d	e	f
a	0	1	1	0	0	1
b	1	0	1	1	0	0
c	1	1	0	1	0	1
d	0	1	1	0	1	0
e	0	0	0	1	0	1
f	1	0	1	0	1	0

×

	a	b	c	d	e	f
a	0	1	1	0	0	1
b	1	0	1	1	0	0
c	1	1	0	1	0	1
d	0	1	1	0	1	0
e	0	0	0	1	0	1
f	1	0	1	0	1	0

- Let  $A=D^2$ .  $A_{ij}$  gives the number of 2-edge paths from node  $i$  to node  $j$
- This works for both directed and undirected graphs! (but not weighted)



# Some Assignment 1 Feedback

- The main issues were not with programming capabilities, but with reading comprehension and double-checking answers.

## Submission Instructions

Follow the steps below:

1. Create a folder and name it as <student no>\_<your name>, e.g., **12345678d\_CHANTaiMan**
2. For Q1, Q2, and Q3, type your answers in a word document and save it as a **.pdf** file. Name the single **.pdf** file as A1\_<student no>\_<your name>.**pdf**, e.g., **A1\_12345678d\_CHANTaiMan.pdf**
3. For Q2 and Q3, submit the source files (**.py**). Name the **.py** files as A1\_Q<question no>\_<student no>\_<your name>.**py**, e.g., **A1\_Q2\_12345678d\_CHANTaiMan.py**
4. Put all the **.pdf** and **.py** files into the folder created in Step 1.
5. Compress the folder (**.zip**, **.7z**, or **.rar**).
6. Submit the file to Blackboard.

A maximum of **3 attempts** for submission are allowed. **Only the last attempt will be graded.** A late penalty of 20% per day will be imposed.

**A corrupted file will be given ZERO marks.** It is your obligation to check carefully the files in your submission.

# Some Assignment 1 Feedback

- The main issues were not with programming capabilities, but with reading comprehension and double-checking answers.
- After submitting, download your submission and make sure there are no issues with it.
- Actually use the test cases we gave you to make sure that your code is correct.

# Some Assignment 1 Feedback

- Please don't write your written answers (pseudocode, efficiency calculations) as comments in the Python code.
- The programmer's brain, like the compiler, is automatically trained to ignore comments.
  - Comments often represent deleted or tentative code.
  - Comments are often used to explain code, so we only read them when we don't understand the code.

# Some Quiz 1 Feedback

**DO NOT TURN THIS PAGE UNTIL ANNOUNCED**

- Assignment 2 will be released soon
  - Due in Week 12
- Quiz 2 in the Flipped Learning lecture next week

