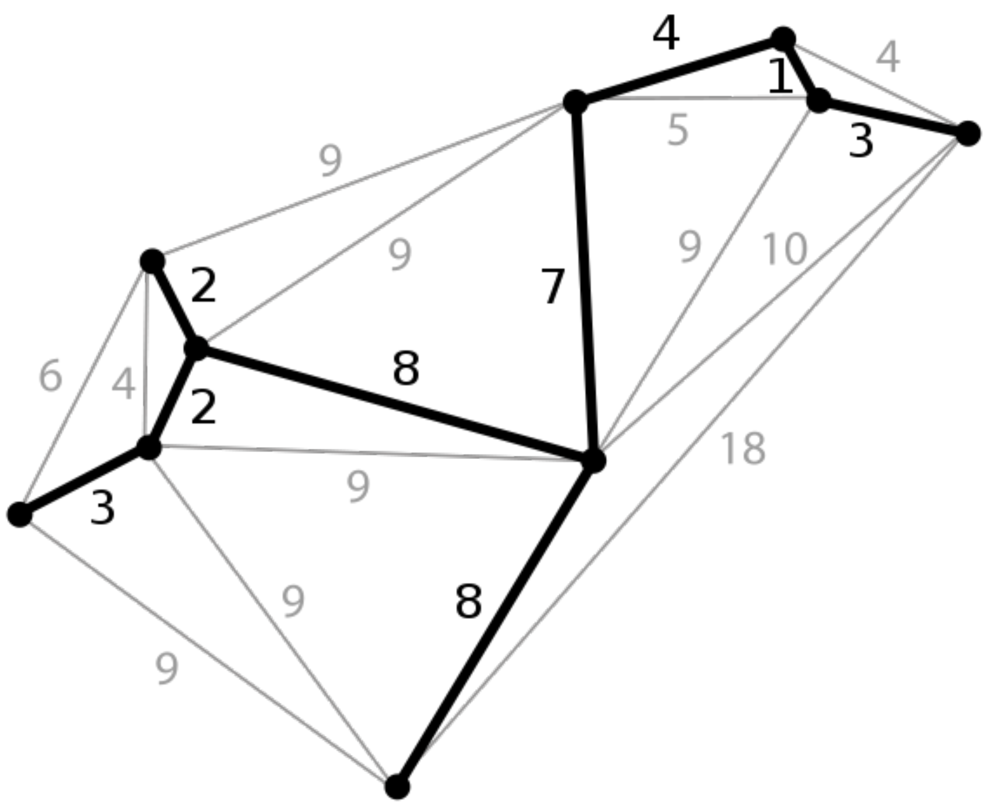
CS 106X

Lecture 23: Graphs II

Monday, March 6, 2017



Programming Abstractions (Accelerated)
Winter 2017
Stanford University
Computer Science Department

Lecturer: Chris Gregg

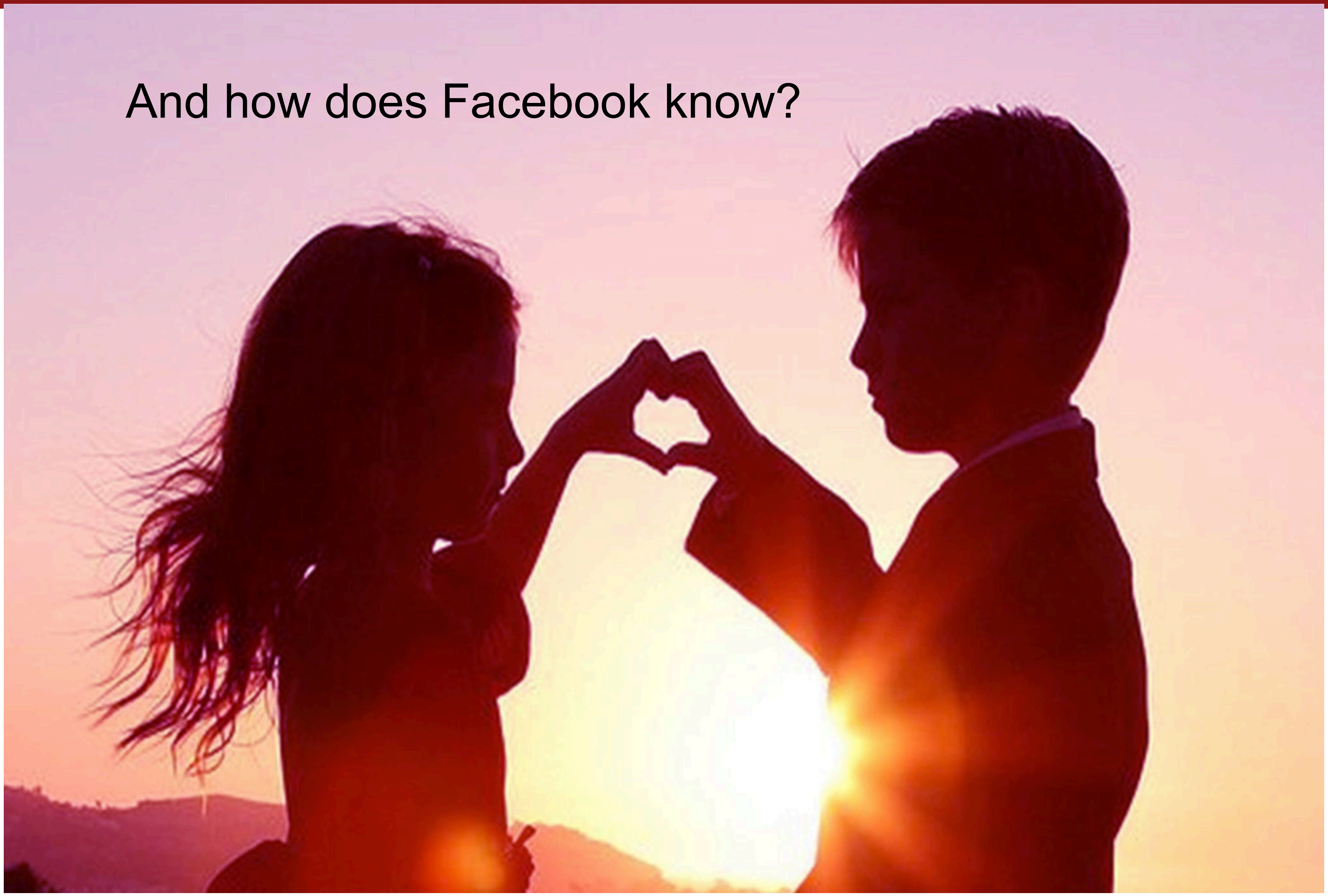
reading:
Programming Abstractions in C++, Chapter 18

Today's Topics

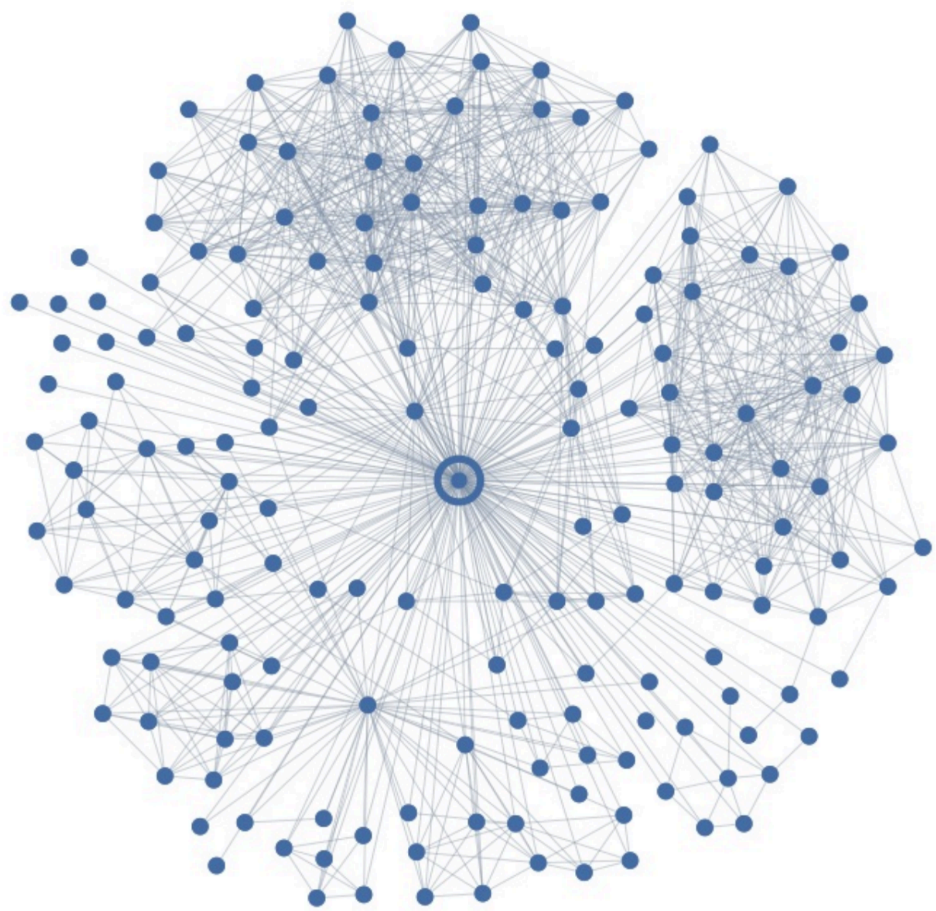
- Logistics

- Finish up Who Do You Love?
- Real Graph: Internet routers and traceroute
- More on Trailblazer
- Minimum Spanning Trees
  - Kruskal's algorithm

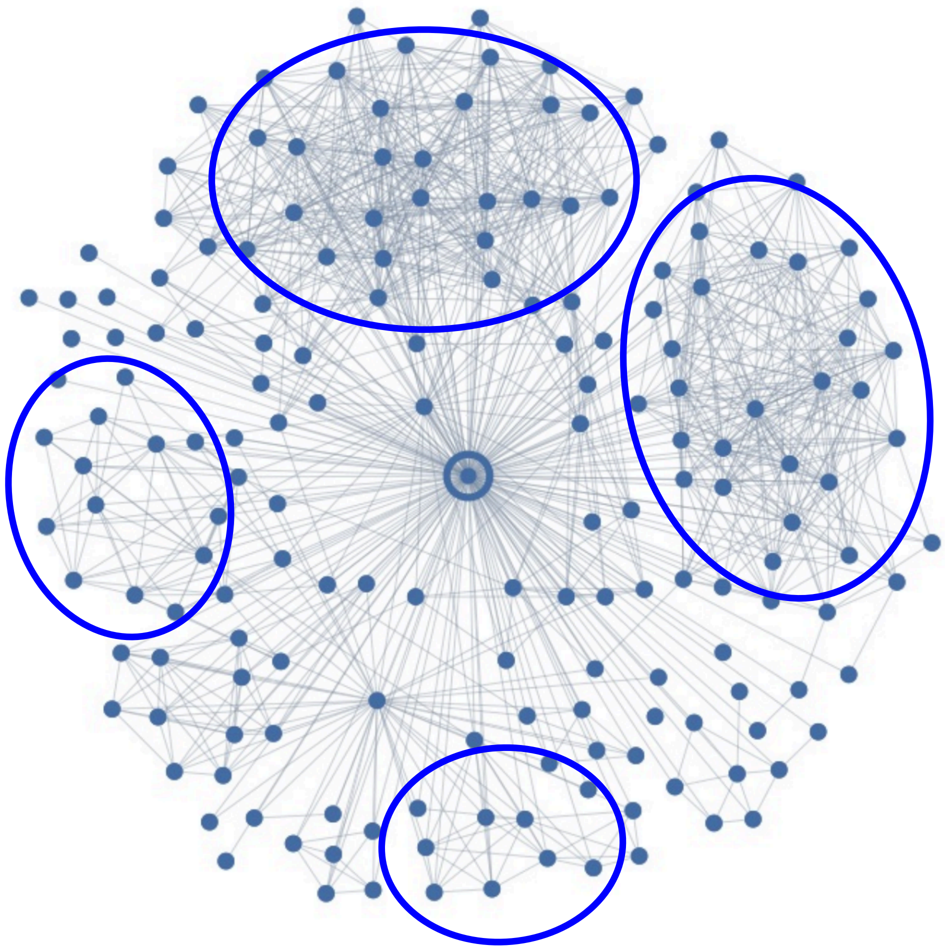
Who Do You Love



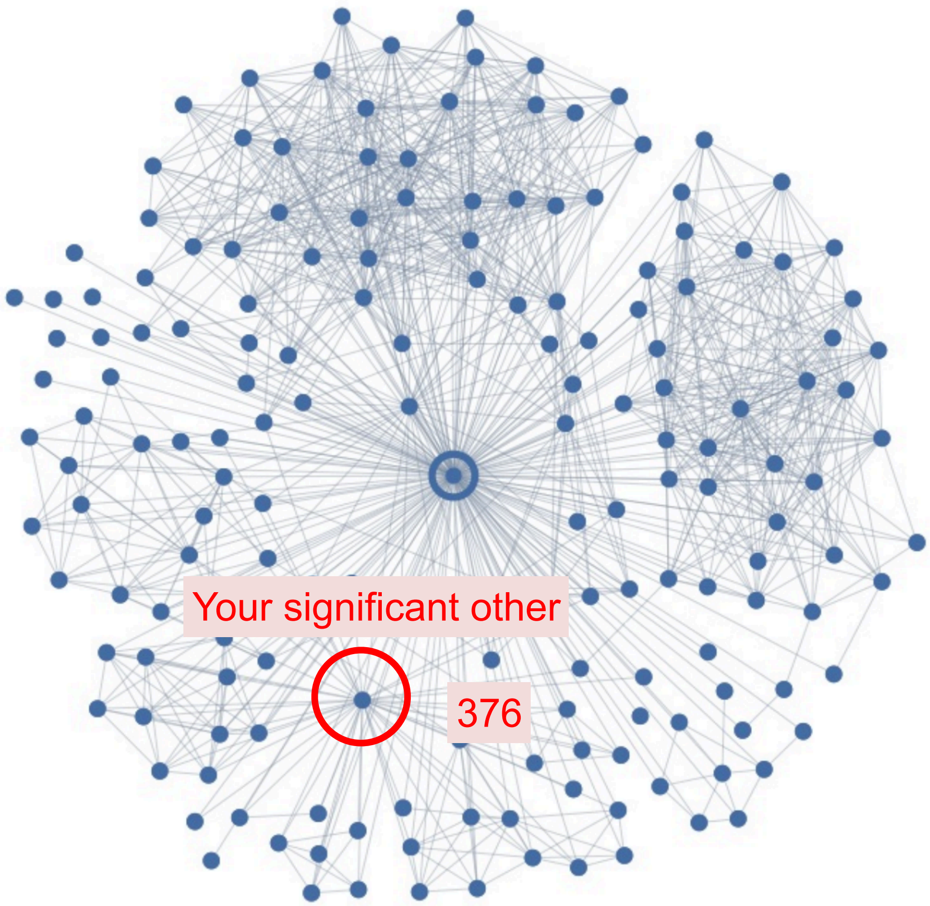
Ego Graph



Maybe I Love These People?



But I Actually Love This Person



Romance and Dispersion

Romantic Partnerships and the Dispersion of Social Ties: A Network Analysis of Relationship Status on Facebook

Lars Backstrom
Facebook Inc.

Jon Kleinberg
Cornell University

ABSTRACT

A crucial task in the analysis of on-line social-networking systems is to identify important people — those linked by strong social ties — within an individual's network neighborhood. Here we investigate this question for a particular category of strong ties, those involving spouses or romantic partners. We organize our analysis around a basic question: given all the connections among a person's friends, can you recognize his or her romantic partner from the network structure alone? Using data from a large sample of Facebook users, we find that this task can be accomplished with high accuracy, but doing so requires the development of a new measure of tie strength that we term 'dispersion' — the extent to which two people's mutual friends are not themselves well-connected. The results offer methods for identifying types of structurally significant people in on-line applications, and suggest a potential expansion of existing theories of tie strength.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database applications—*Data mining*
Keywords: Social Networks; Romantic Relationships.

they see from friends [1], and organizing their neighborhood into conceptually coherent groups [23, 25].

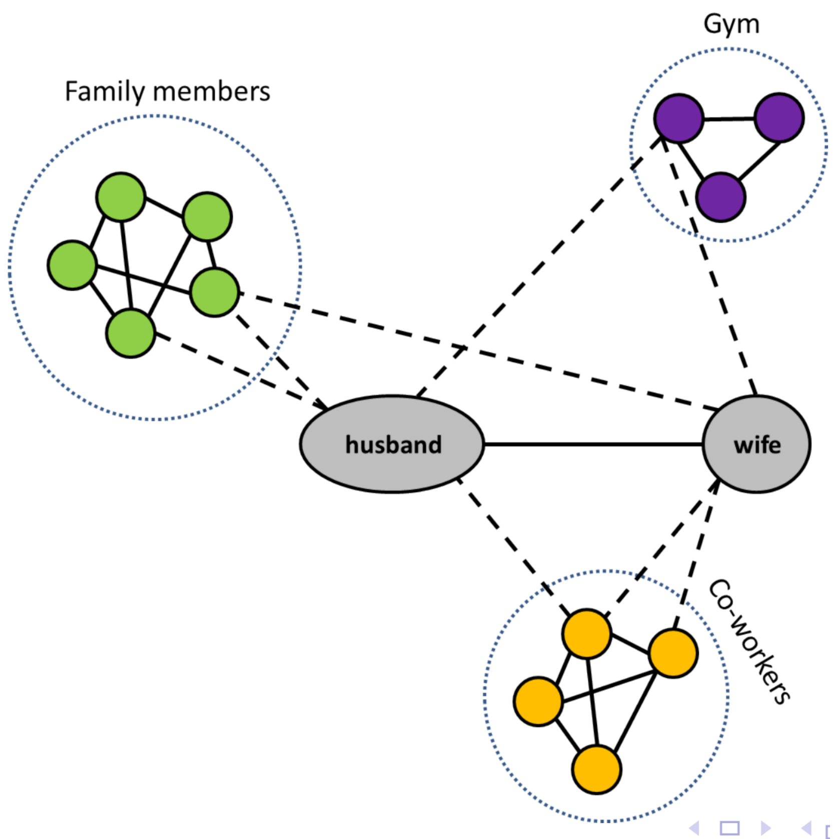
Tie Strength.
Tie strength forms an important dimension along which to characterize a person's links to their network neighbors. Tie strength informally refers to the 'closeness' of a friendship; it captures a spectrum that ranges from strong ties with close friends to weak ties with more distant acquaintances. An active line of research reaching back to foundational work in sociology has studied the relationship between the strengths of ties and their structural role in the underlying social network [15]. Strong ties are typically 'embedded' in the network, surrounded by a large number of mutual friends [6, 16], and often involving large amounts of shared time together [22] and extensive interaction [17]. Weak ties, in contrast, often involve few mutual friends and can serve as 'bridges' to diverse parts of the network, providing access to novel information [5, 15].

A fundamental question connected to our und
strong ties is to identify the
person's social network

October 2013

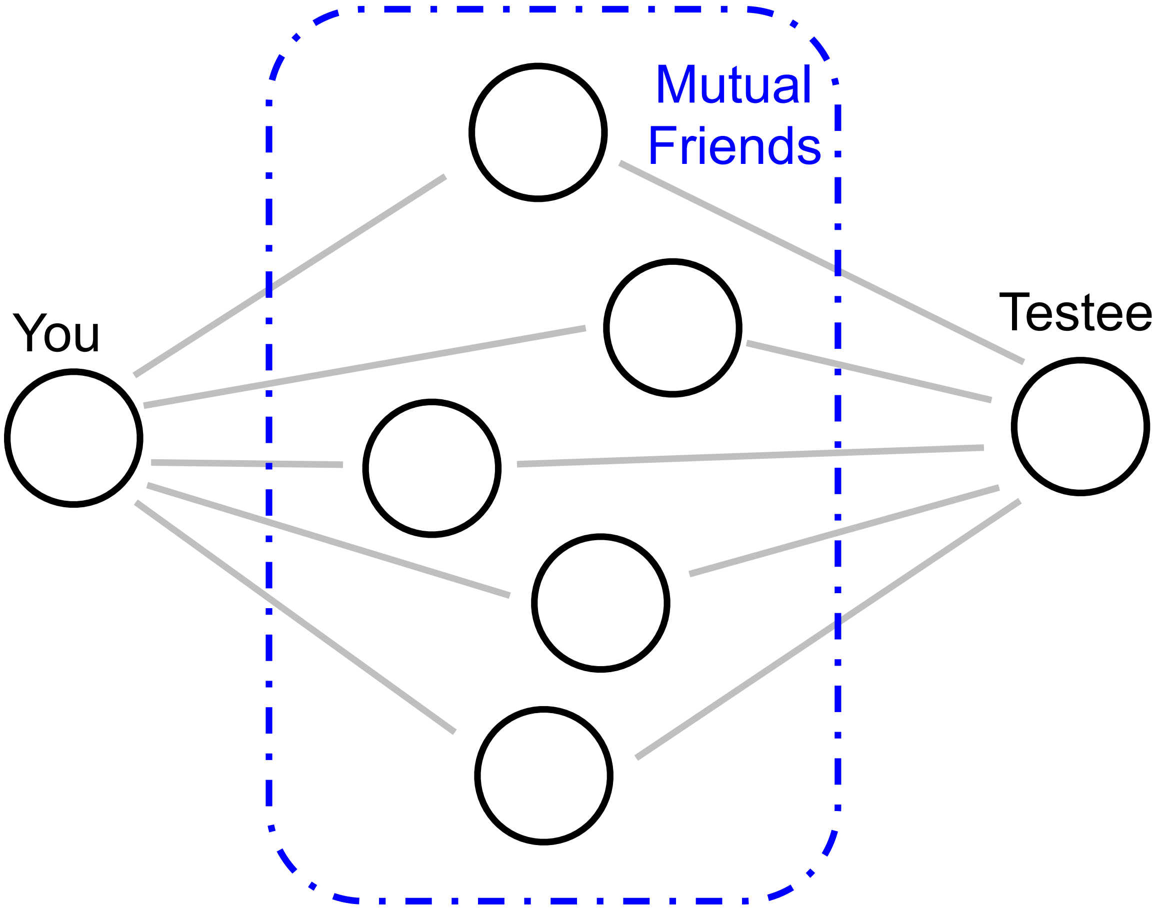
http://arxiv.org/pdf/1310.6753v1.pdf

Dispersion Insight



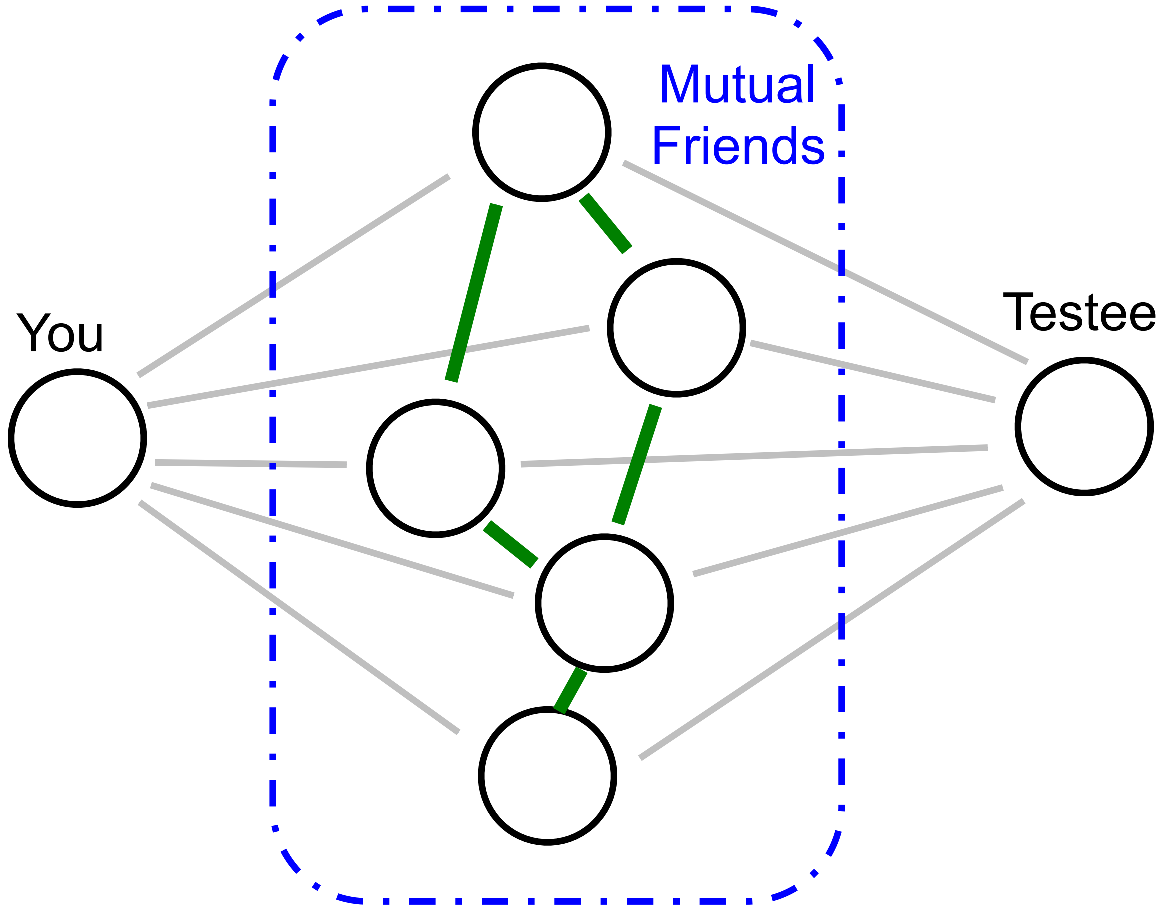
Dispersion: The extent to which two people's mutual friends are not directly connected

Dispersion



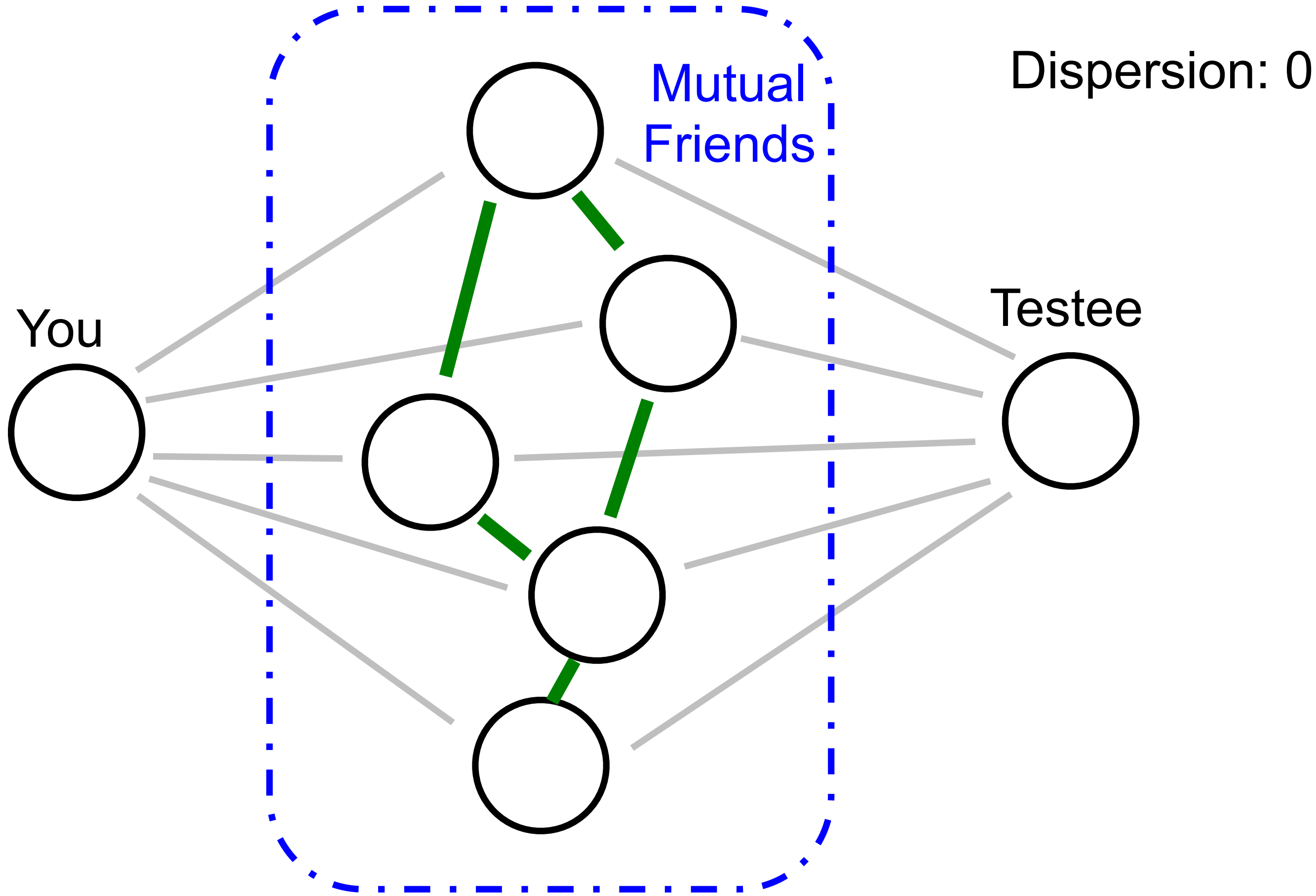
Dispersion: The extent to which two people's mutual friends are not directly connected

Dispersion



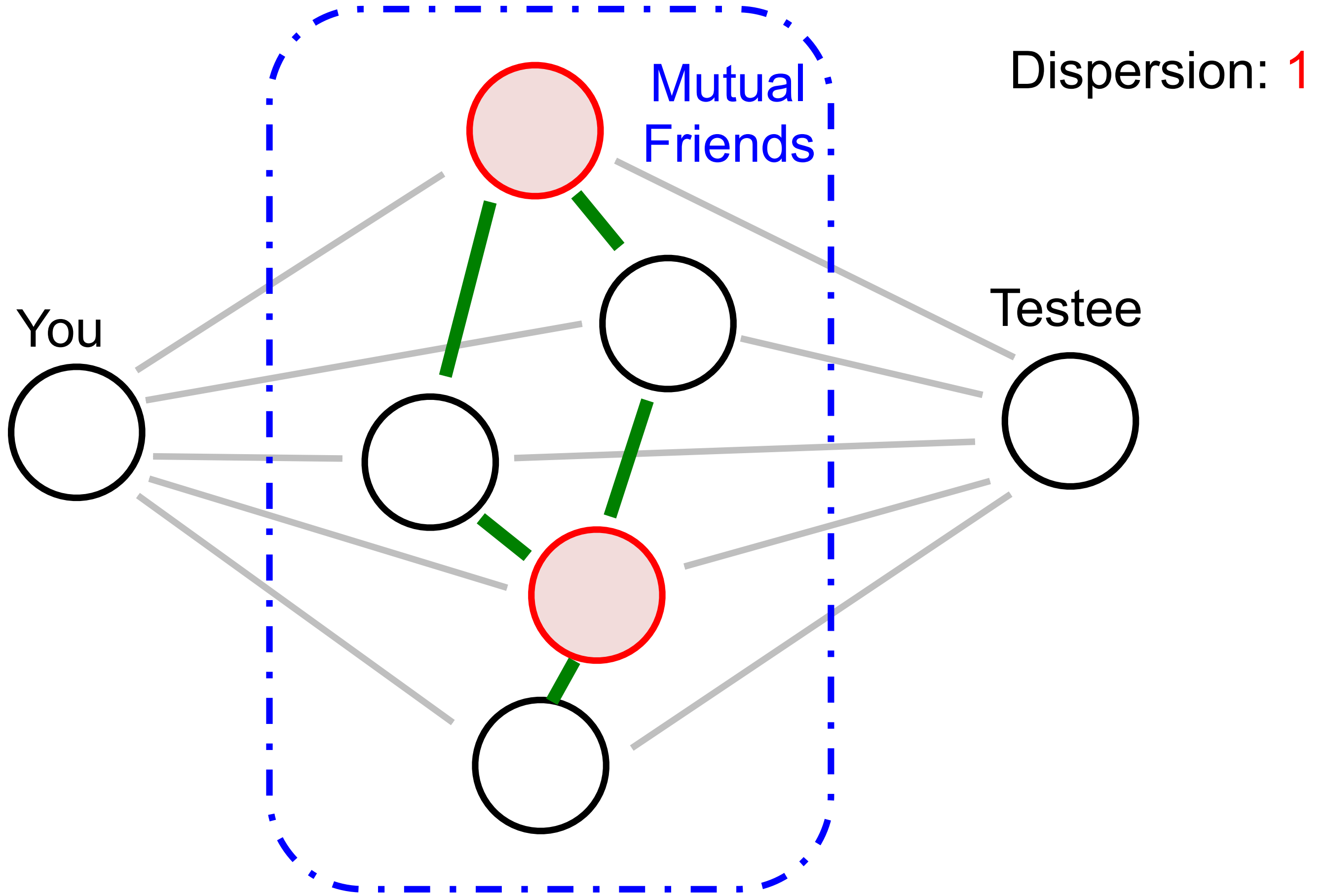
Dispersion: The extent to which two people's mutual friends are not directly connected

Dispersion



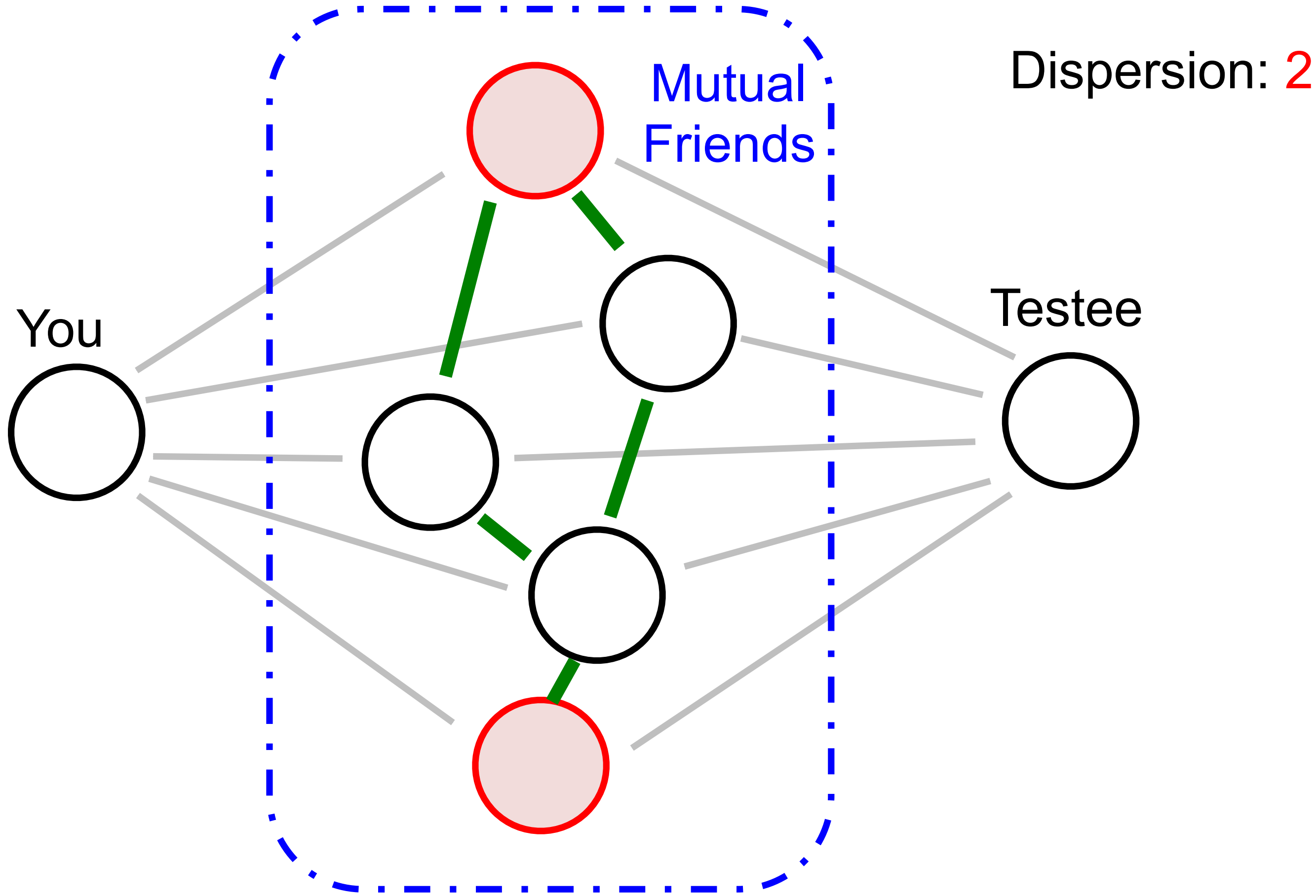
Dispersion: The extent to which two people's mutual friends are not directly connected

Dispersion



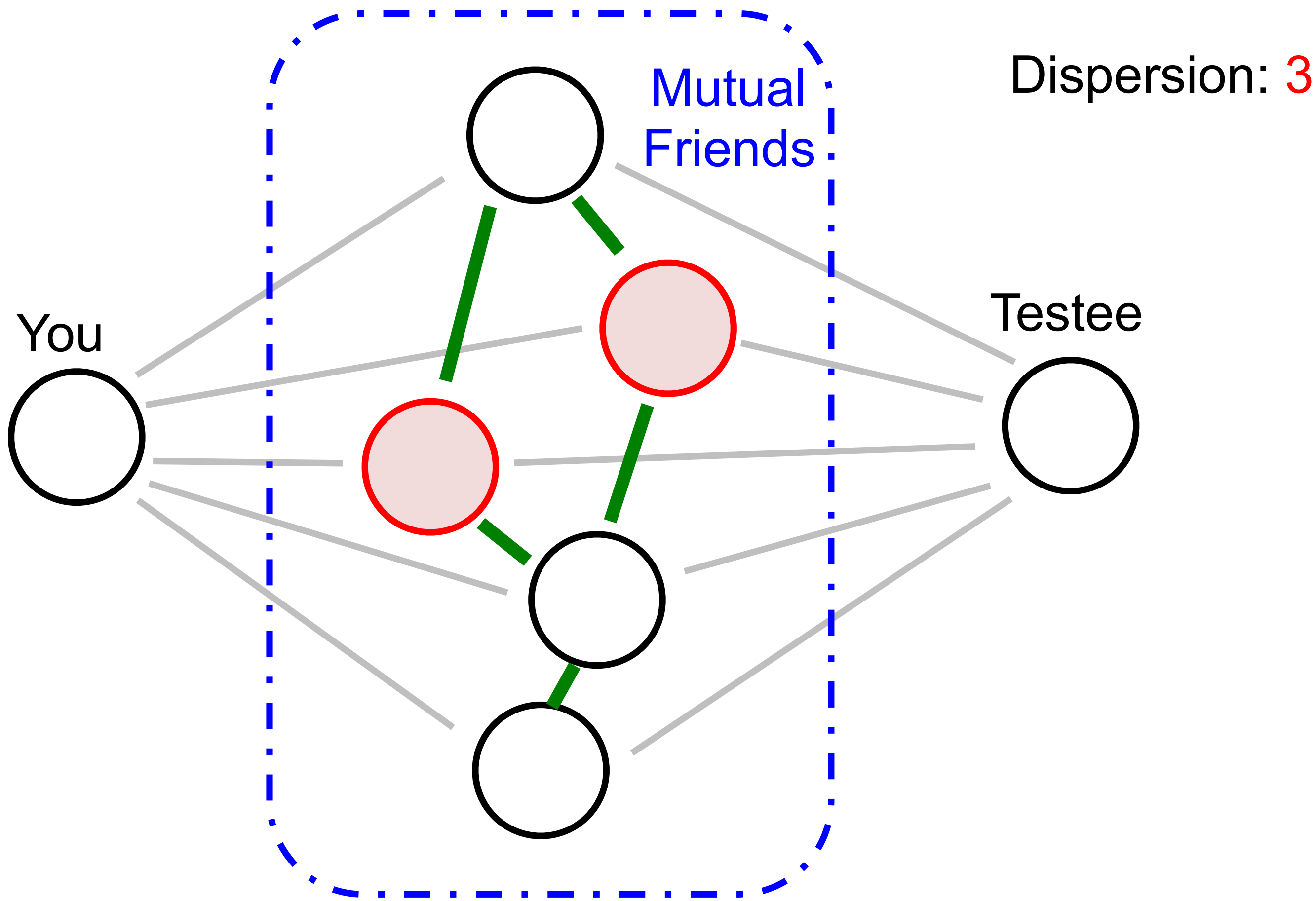
Dispersion: The extent to which two people's mutual friends are not directly connected

Dispersion



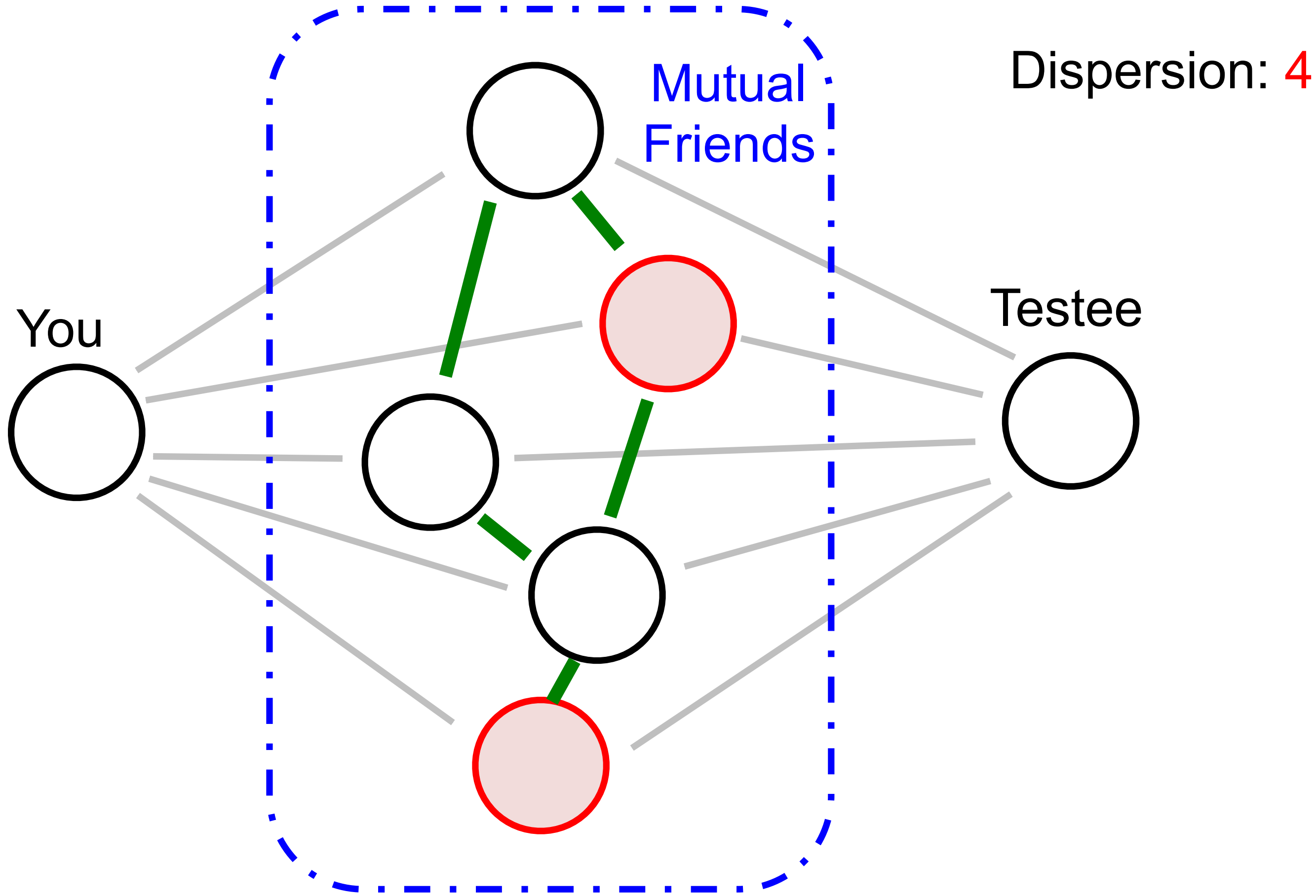
Dispersion: The extent to which two people's mutual friends are not directly connected

Dispersion



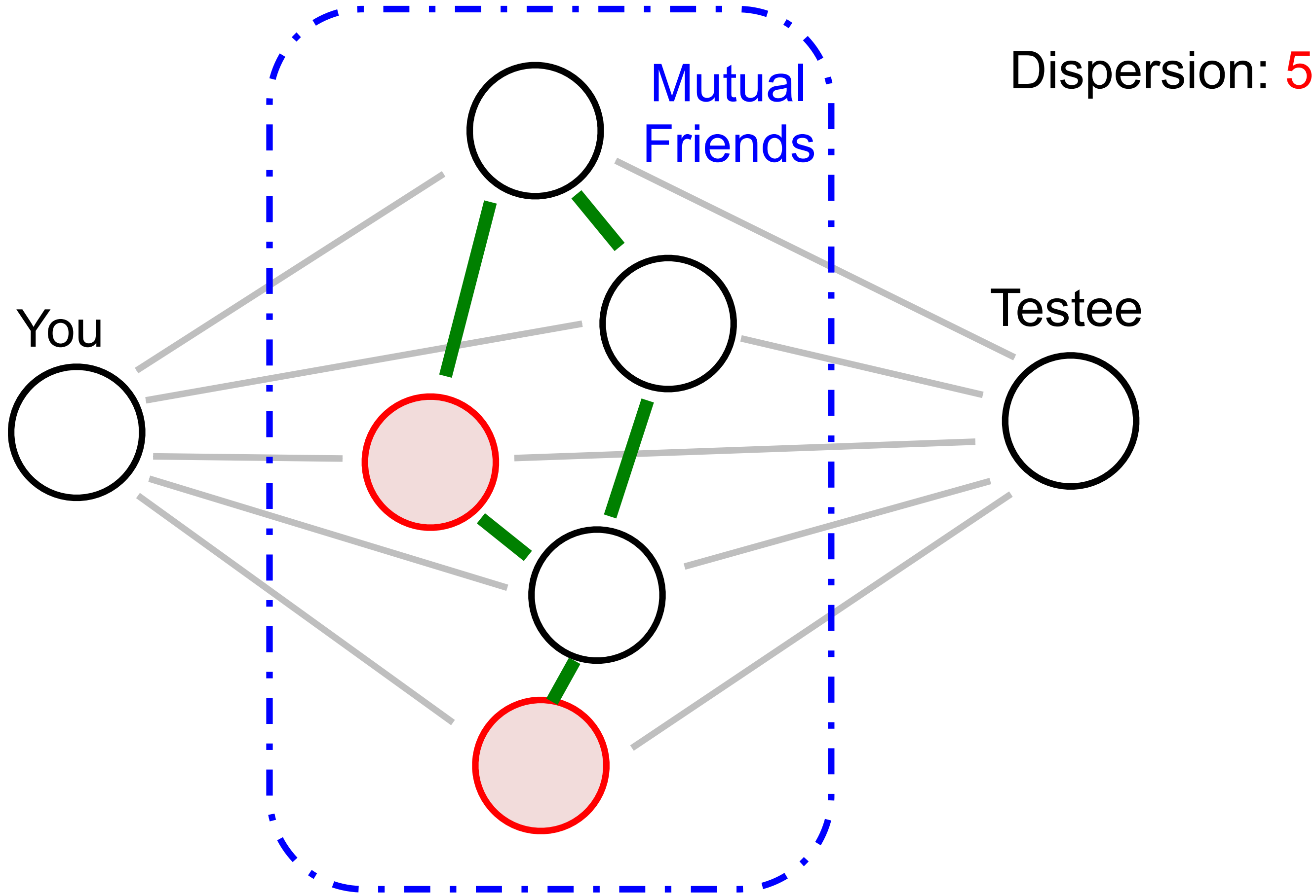
Dispersion: The extent to which two people's mutual friends are not directly connected

Dispersion



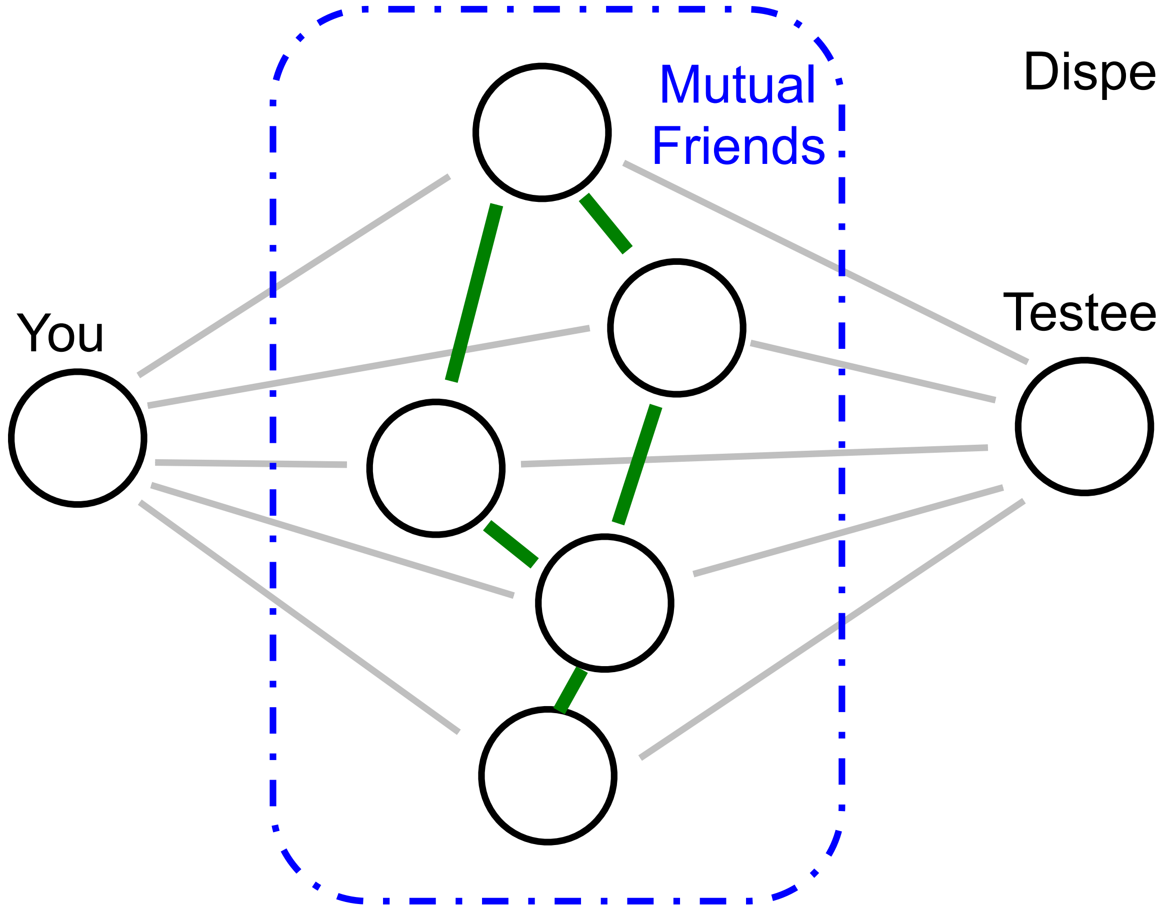
Dispersion: The extent to which two people's mutual friends are not directly connected

Dispersion



Dispersion: The extent to which two people's mutual friends are not directly connected

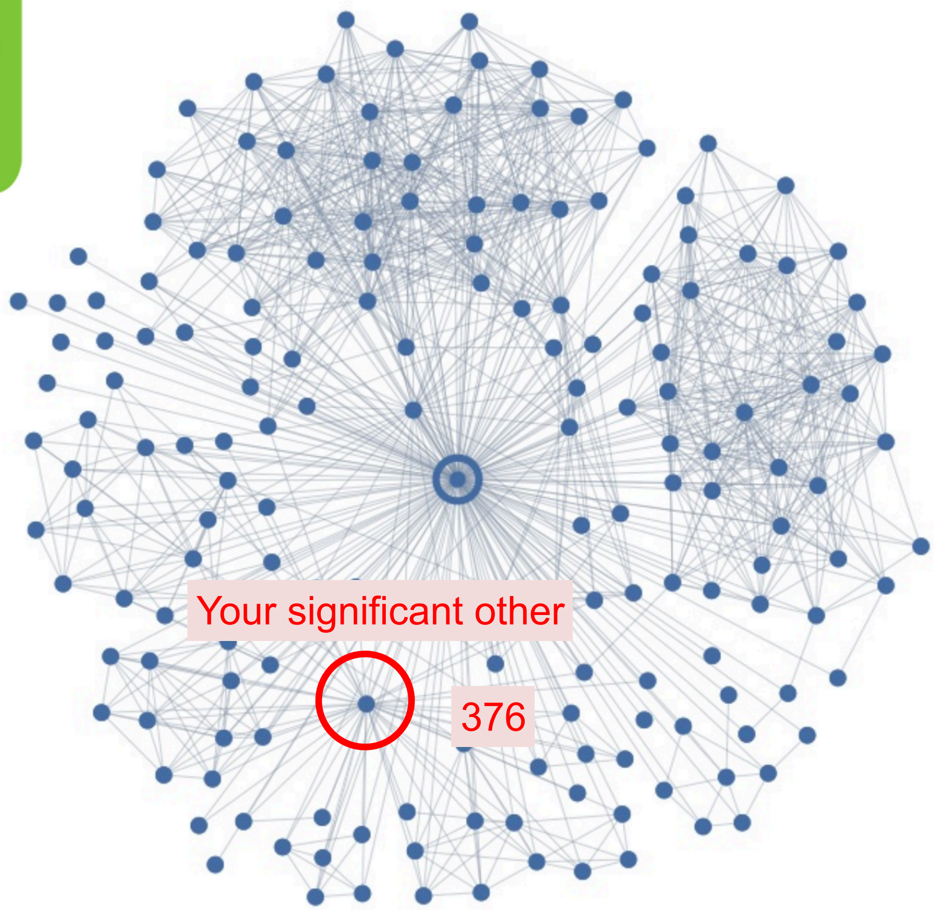
Dispersion



Dispersion: 5

Dispersion: The extent to which two people's mutual friends are not directly connected

Who Do You Love?



Real Graphs!

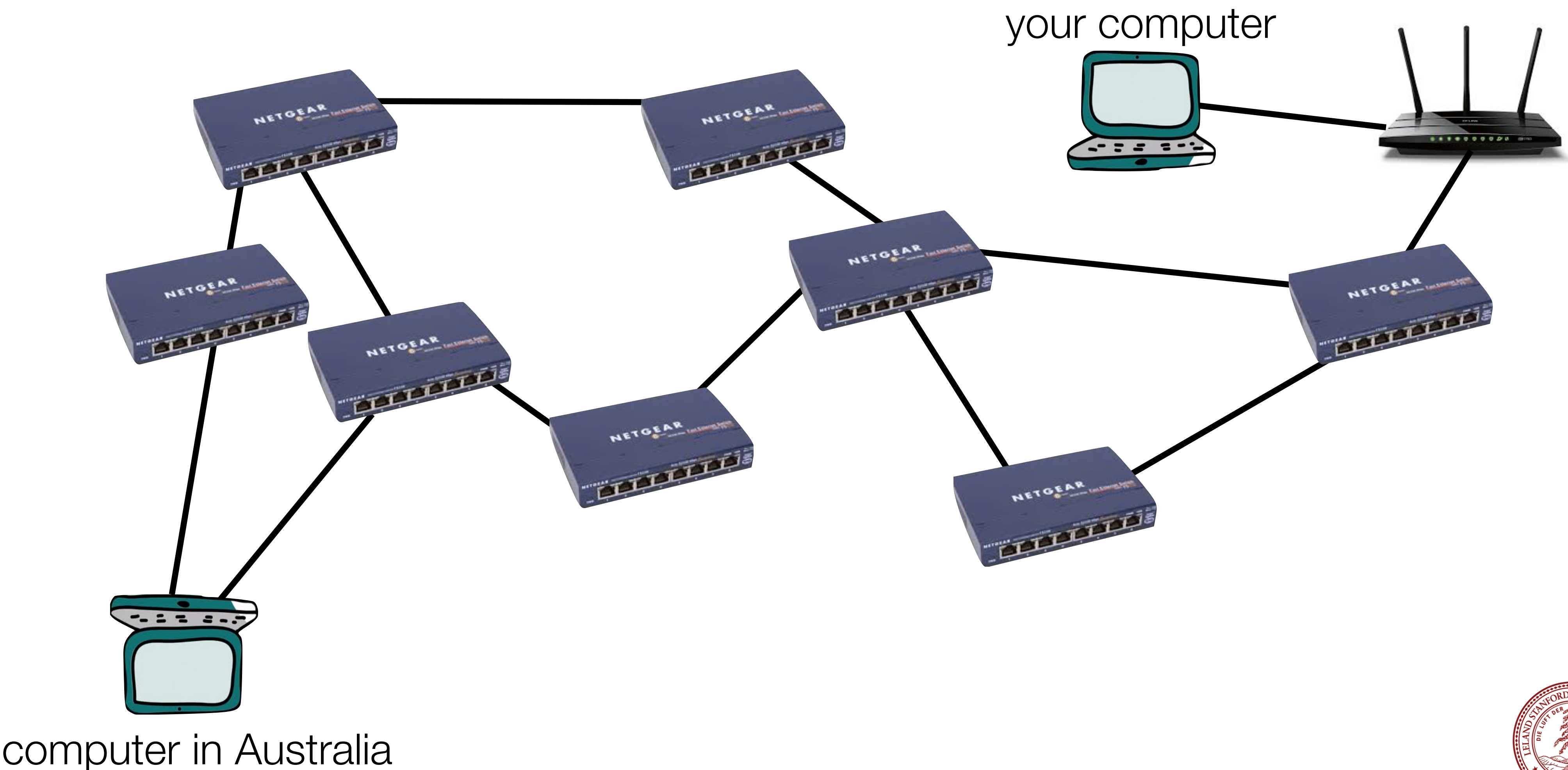
There was a Tiny Feedback from the last lecture that said,

> "*All the different real life examples of graphs made it very interesting*"

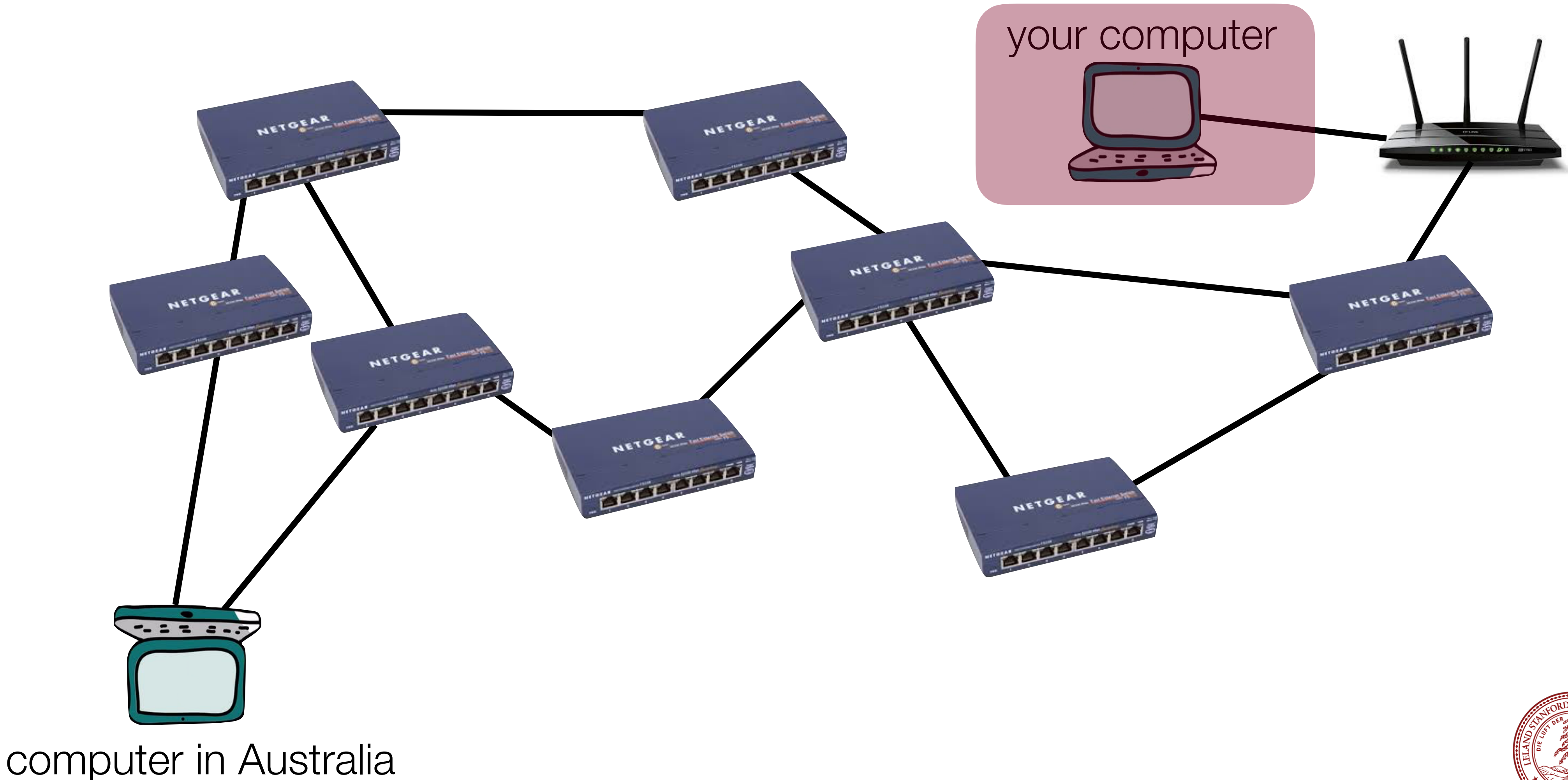
Let's dig a bit deeper into how the Internet is a real graph by analyzing internet routers, or:

How does a message get sent from your computer to another computer on the Internet, say in Australia?

The Internet: Computers connected through routers



The Internet: Computers connected through routers



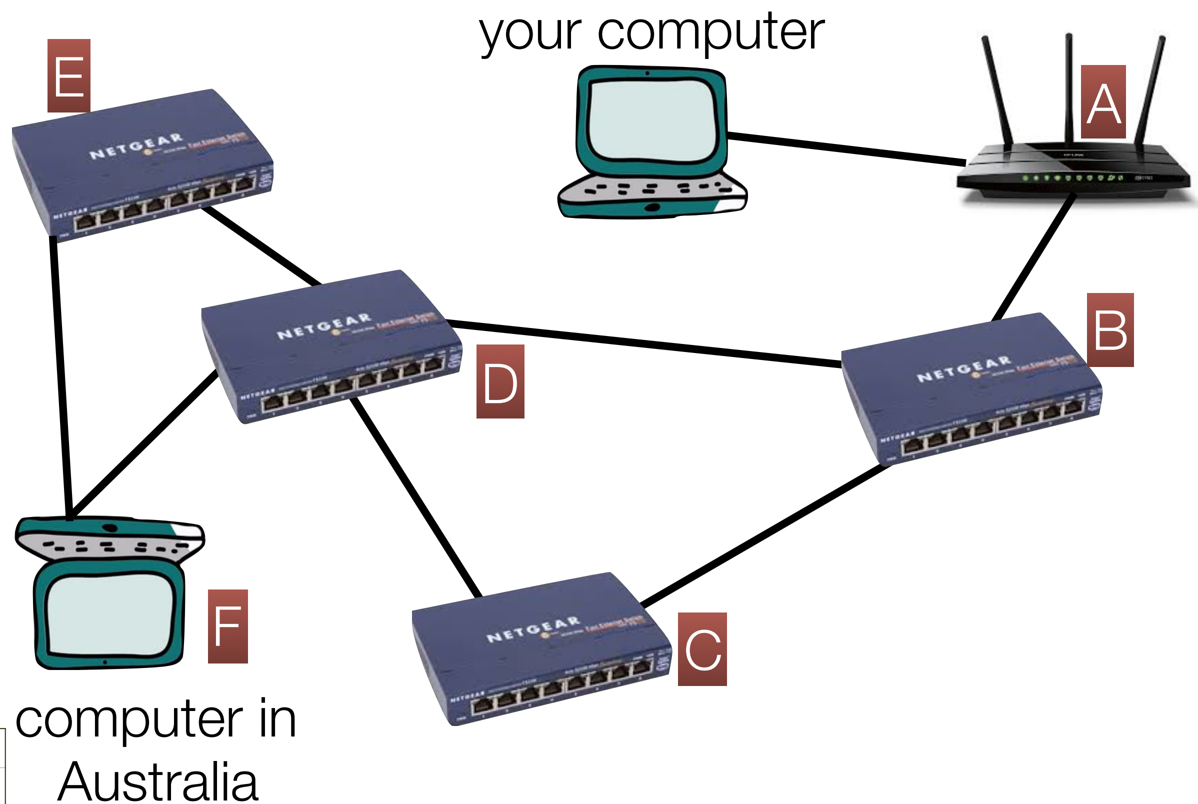
The Internet: Let's simplify a bit

The destination computer has a name and an IP address, like this:

`www.engineering.unsw.edu.au`
`IP address: 149.171.158.109`

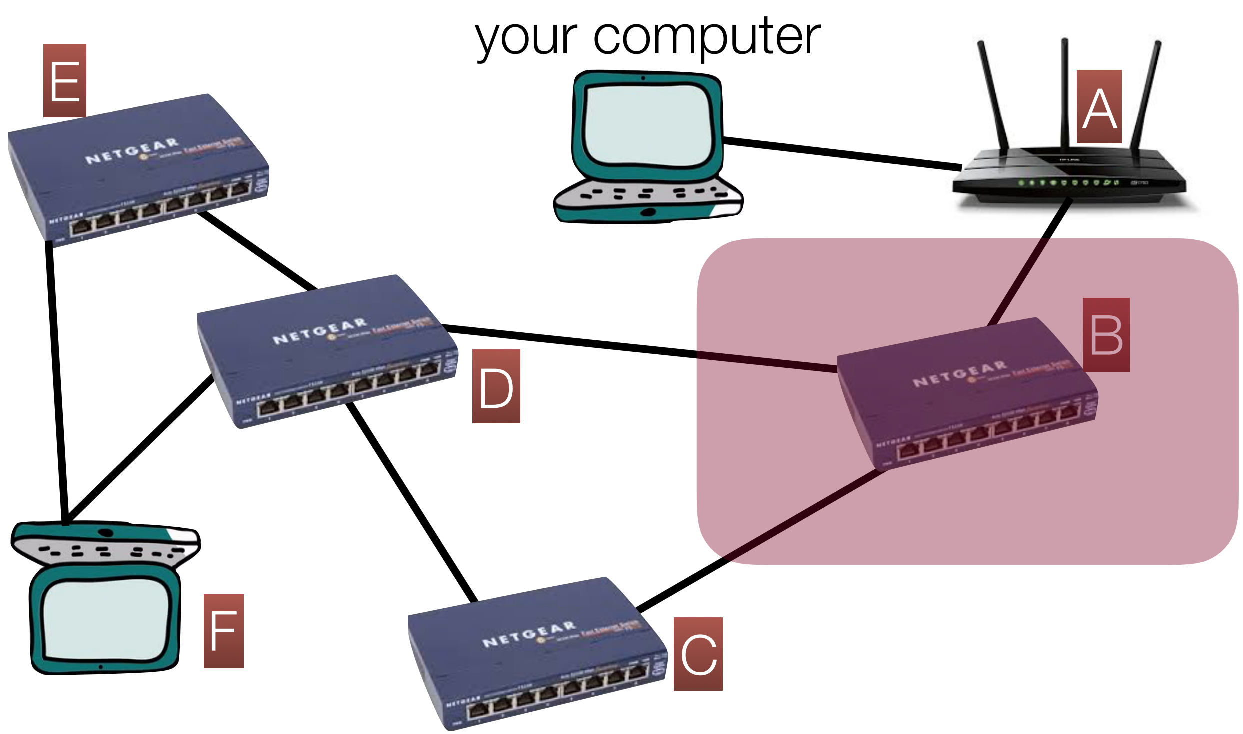
The first number denotes the "network address" and routers continually pass around information about how many "hops" they think it will take for them to get to all the networks. E.g., for router `C`:

router	hops
A	2
B	1
C	-
D	1
E	2
F	2



The Internet: Let's simplify a bit

Each router knows its neighbors, and it has a copy of its neighbors' tables. So, `B` would have the following tables:



A

router	hops
A	-
B	1
C	3
D	2
E	3
F	3

C

router	hops
A	2
B	1
C	-
D	1
E	2
F	2

D

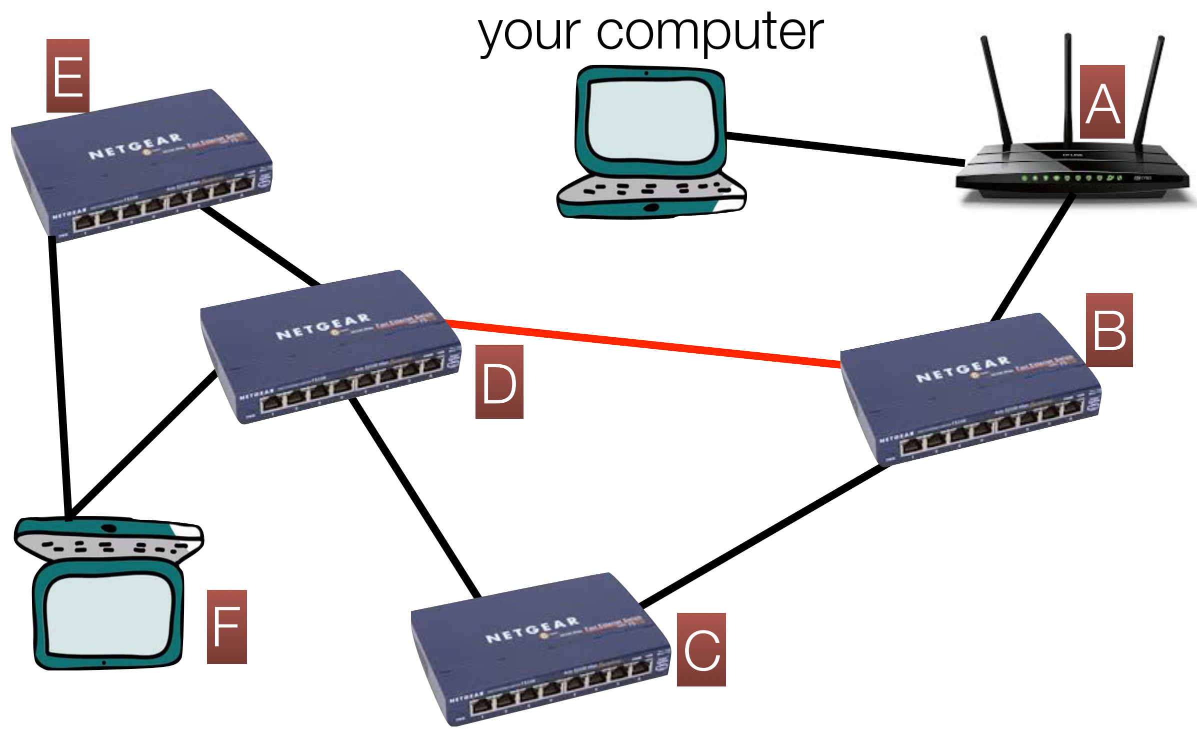
router	hops
A	2
B	1
C	1
D	-
E	1
F	1

The Internet: Let's simplify a bit

If B wants to connect to F, it connects through its neighbor that reports the shortest path to F. Which router would it choose?

A

router	hops
A	-
B	1
C	3
D	2
E	3
F	3



C

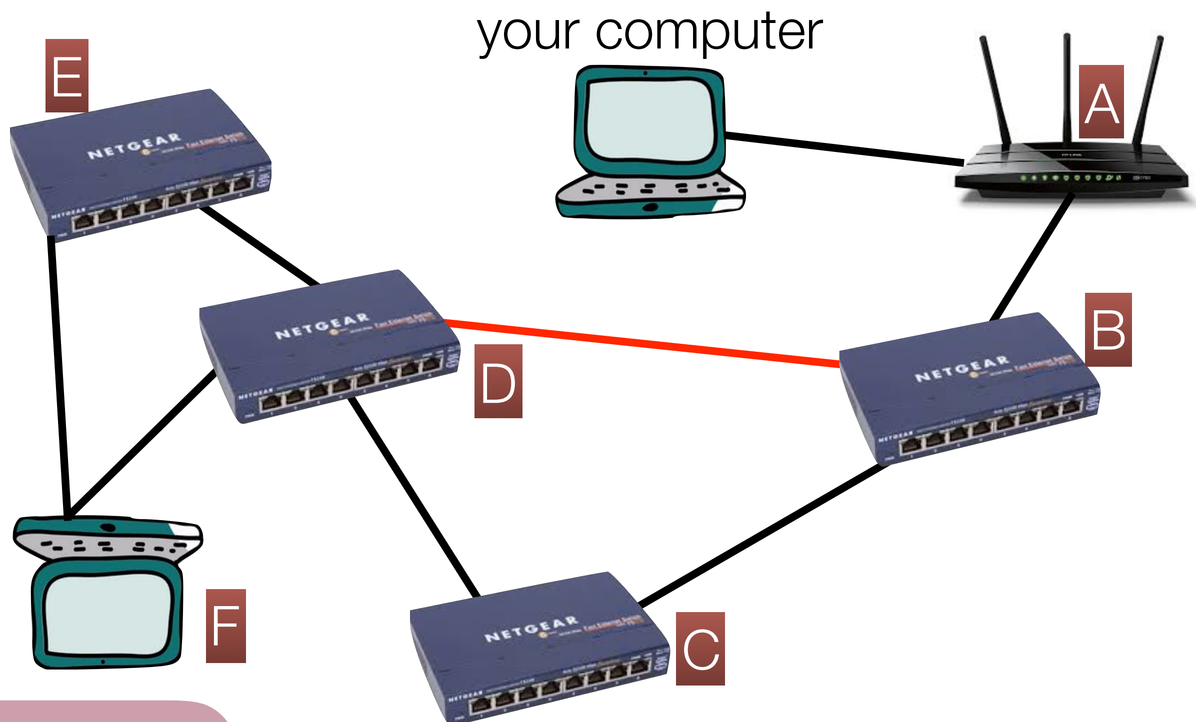
router	hops
A	2
B	1
C	-
D	1
E	2
F	2

D

router	hops
A	2
B	1
C	1
D	-
E	1
F	1

The Internet: Let's simplify a bit

If B wants to connect to F, it connects through its neighbor that reports the shortest path to F. Which router would it choose? D.



A

router	hops
A	-
B	1
C	3
D	2
E	3
F	3

C

router	hops
A	2
B	1
C	-
D	1
E	2
F	2

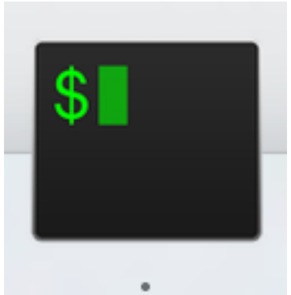
D

router	hops
A	2
B	1
C	1
D	-
E	1
F	1

Traceroute

We can use a program called "traceroute" to tell us the path between our computer and a different computer:

`traceroute -I -e www.engineering.unsw.edu.au`



Traceroute: Stanford Hops

```
traceroute -I -e www.engineering.unsw.edu.au
traceroute to www.engineering.unsw.edu.au (149.171.158.109), 64 hops max, 72 byte packets
 1  csmx-west-rtr.sunet (171.67.64.2)  7.414 ms  9.155 ms  8.288 ms
 2  gnat-2.sunet (172.24.70.12)  0.339 ms  1.532 ms  0.423 ms
 3  csmx-west-rtr-vl3866.sunet (171.64.66.2)  38.916 ms  10.506 ms  8.402 ms
 4  dca-rtr-vlan8.sunet (171.64.255.204)  0.530 ms  0.521 ms  0.713 ms
 5  dc-svl-agg4--stanford-10ge.cenic.net (137.164.50.157)  1.554 ms  1.653 ms  2.828 ms
 6  hpr-svl-hpr2--svl-agg4-10ge.cenic.net (137.164.26.249)  1.212 ms  1.161 ms  1.204 ms
 7  aarnet-2-is-jmb-778.sttlwa.pacificwave.net (207.231.245.4)  17.994 ms  17.998 ms  18.319 ms
 8  et-2-0-0.pe2.brwy.nsw.aarnet.net.au (113.197.15.98)  160.020 ms  160.234 ms  159.922 ms
 9  et-3-3-0.pe1.brwy.nsw.aarnet.net.au (113.197.15.148)  160.285 ms  160.076 ms  160.118 ms
10  138.44.5.1 (138.44.5.1)  160.124 ms  160.138 ms  160.068 ms
11  ombcr1-te-1-5.gw.unsw.edu.au (149.171.255.106)  160.090 ms  160.381 ms  160.185 ms
12  r1dcdnex1-po-2.gw.unsw.edu.au (149.171.255.178)  160.909 ms  160.847 ms  160.921 ms
13  dcfw1-ae-1-3049.gw.unsw.edu.au (129.94.254.60)  160.592 ms  160.558 ms  160.949 ms
14  www.engineering.unsw.edu.au (149.171.158.109)  160.978 ms  161.184 ms  160.987 ms
```

Traceroute: CENIC

```
traceroute -I -e www.engineering.unsw.edu.au
traceroute to www.engineering.unsw.edu.au (149.171.158.109), 64 hops max, 72 byte packets
 1  csmx-west-rtr.sunet (171.67.64.2)  7.414 ms  9.155 ms  8.288 ms
 2  gnat-2.sunet (172.24.70.12)  0.339 ms  1.532 ms  0.423 ms
 3  csmx-west-rtr-vl3866.sunet (171.64.66.2)  38.916 ms  10.506 ms  8.402 ms
 4  dca-rtr-vlan8.sunet (171.64.255.204)  0.530 ms  0.521 ms  0.713 ms
 5  dc-svl-agg4--stanford-10ge.cenic.net (137.164.50.157)  1.554 ms  1.653 ms  2.828 ms
 6  hpr-svl-hpr2--svl-agg4-10ge.cenic.net (137.164.26.249)  1.212 ms  1.161 ms  1.204 ms
 7  aarnet-2-is-jmb-778.sttlwa.pacificwave.net (207.231.245.4)  17.994 ms  17.998 ms  18.319 ms
 8  et-2-0-0.pe2.brwy.nsw.aarnet.net.au (113.197.15.98)  160.020 ms  160.234 ms  159.922 ms
 9  et-3-3-0.pe1.brwy.nsw.aarnet.net.au (113.197.15.148)  160.285 ms  160.076 ms  160.118 ms
10  138.44.5.1 (138.44.5.1)  160.124 ms  160.138 ms  160.068 ms
11  ombcr1-te-1-5.gw.unsw.edu.au (149.171.255.106)  160.090 ms  160.381 ms  160.185 ms
12  r1dcdnex1-po-2.gw.unsw.edu.au (149.171.255.178)  160.909 ms  160.847 ms  160.921 ms
13  dcfw1-ae-1-3049.gw.unsw.edu.au (129.94.254.60)  160.592 ms  160.558 ms  160.949 ms
14  www.engineering.unsw.edu.au (149.171.158.109)  160.978 ms  161.184 ms  160.987 ms
```

The **Corporation for Education Network Initiatives in California** (**CENIC**) is a nonprofit corporation formed in 1996 to provide high-performance, high-bandwidth networking services to California universities and research institutions (source: Wikipedia)

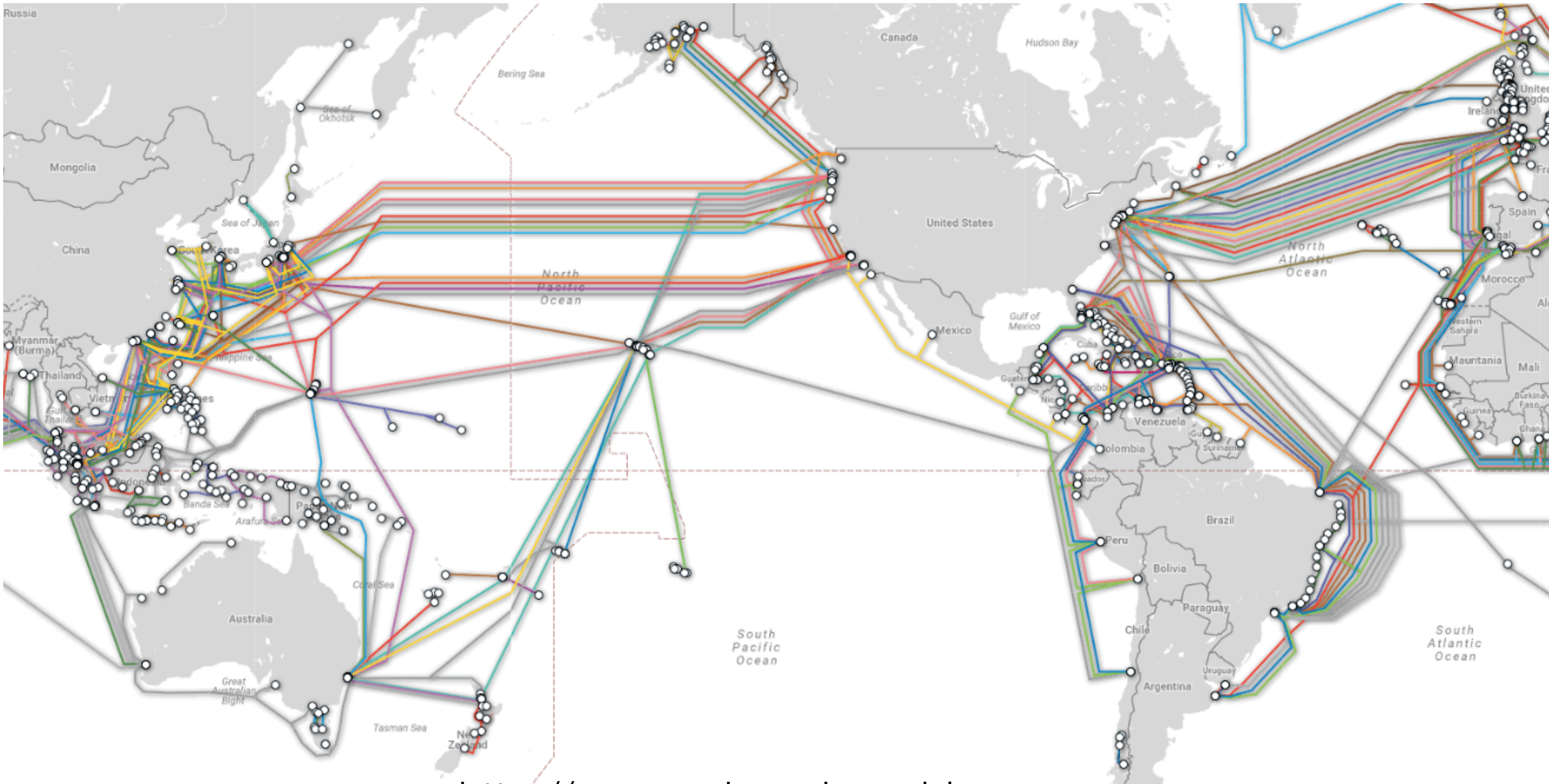
Traceroute: Pacificwave (Seattle)

```
traceroute -I -e www.engineering.unsw.edu.au
traceroute to www.engineering.unsw.edu.au (149.171.158.109), 64 hops max, 72 byte packets
 1  csmx-west-rtr.sunet (171.67.64.2)  7.414 ms  9.155 ms  8.288 ms
 2  gnat-2.sunet (172.24.70.12)  0.339 ms  1.532 ms  0.423 ms
 3  csmx-west-rtr-vl3866.sunet (171.64.66.2)  38.916 ms  10.506 ms  8.402 ms
 4  dca-rtr-vlan8.sunet (171.64.255.204)  0.530 ms  0.521 ms  0.713 ms
 5  dc-svl-agg4--stanford-10ge.cenic.net (137.164.50.157)  1.554 ms  1.653 ms  2.828 ms
 6  hpr-svl-hpr2--svl-agg4-10ge.cenic.net (137.164.26.249)  1.212 ms  1.161 ms  1.204 ms
 7  aarnet-2-is-jmb-778.sttlwa.pacificwave.net (207.231.245.4)  17.994 ms  17.998 ms  18.319 ms
 8  et-2-0-0.pe2.brwy.nsw.aarnet.net.au (113.197.15.98)  160.020 ms  160.234 ms  159.922 ms
 9  et-3-3-0.pe1.brwy.nsw.aarnet.net.au (113.197.15.148)  160.285 ms  160.076 ms  160.118 ms
10  138.44.5.1 (138.44.5.1)  160.124 ms  160.138 ms  160.068 ms
    gw.unsw.edu.au (149.171.255.106)  160.090 ms  160.381 ms  160.185 ms
    .gw.unsw.edu.au (149.171.255.178)  160.909 ms  160.847 ms  160.921 ms
    9.gw.unsw.edu.au (129.94.254.60)  160.592 ms  160.558 ms  160.949 ms
    g.unsw.edu.au (149.171.158.109)  160.978 ms  161.184 ms  160.987 ms
```

PACIFIC NORTHWEST GIGAPOP

Pass Internet traffic directly with other major national and international networks, including U.S. federal agencies and many Pacific Rim R&E networks (source: http://www.pnwgp.net/services/pacific-wave-peering-exchange/)

Traceroute: Oregon to Australia - underwater!



http://www.submarinecablemap.com

Traceroute: Australia

```
traceroute -I -e www.engineering.unsw.edu.au
traceroute to www.engineering.unsw.edu.au (149.171.158.109), 64 hops max, 72 byte packets
 1  csmx-west-rtr.sunet (171.67.64.2)  7.414 ms  9.155 ms  8.288 ms
 2  gnat-2.sunet (172.24.70.12)  0.339 ms  1.532 ms  0.423 ms
 3  csmx-west-rtr-vl3866.sunet (171.64.66.2)  38.916 ms  10.506 ms  8.402 ms
 4  dca-rtr-vlan8.sunet (171.64.255.204)  0.530 ms  0.521 ms  0.713 ms
 5  dc-svl-agg4--stanford-10ge.cenic.net (137.164.50.157)  1.554 ms  1.653 ms  2.828 ms
 6  hpr-svl-hpr2--svl-agg4-10ge.cenic.net (137.164.26.249)  1.212 ms  1.161 ms  1.204 ms
 7  aarnet-2-is-jmb-778.sttlwa.pacificwave.net (207.231.245.4)  17.994 ms  17.998 ms  18.319 ms
 8  et-2-0-0.pe2.brwy.nsw.aarnet.net.au (113.197.15.98)  160.020 ms  160.234 ms  159.922 ms
 9  et-3-3-0.pe1.brwy.nsw.aarnet.net.au (113.197.15.148)  160.285 ms  160.076 ms  160.118 ms
10  138.44.5.1 (138.44.5.1)  160.124 ms  160.138 ms  160.068 ms
11  ombcr1-te-1-5.gw.unsw.edu.au (149.171.255.106)  160.090 ms  160.381 ms  160.185 ms
12  r1dcdnex1-po-2.gw.unsw.edu.au (149.171.255.178)  160.909 ms  160.847 ms  160.921 ms
13  dcfw1-ae-1-3049.gw.unsw.edu.au (129.94.254.60)  160.592 ms  160.558 ms  160.949 ms
14  www.engineering.unsw.edu.au (149.171.158.109)  160.978 ms  161.184 ms  160.987 ms
```

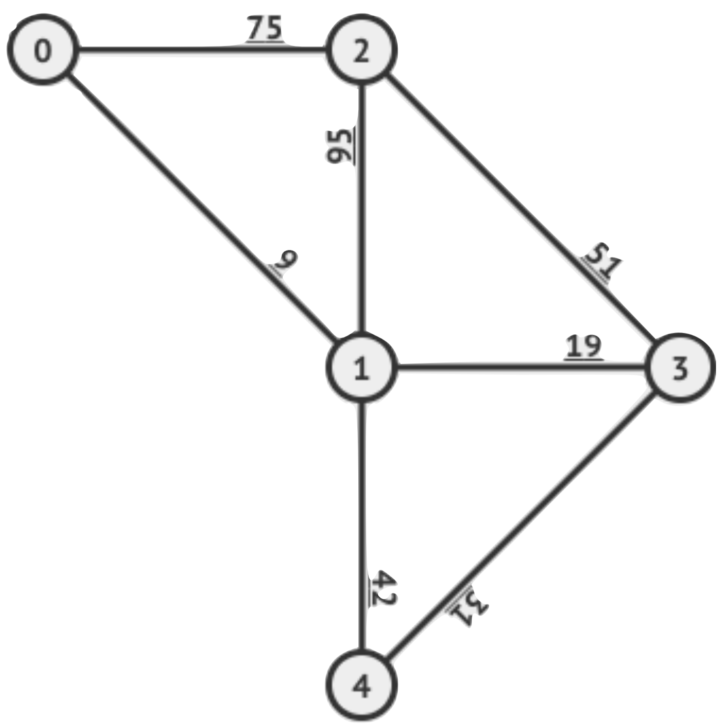
Traceroute: University of New South Wales

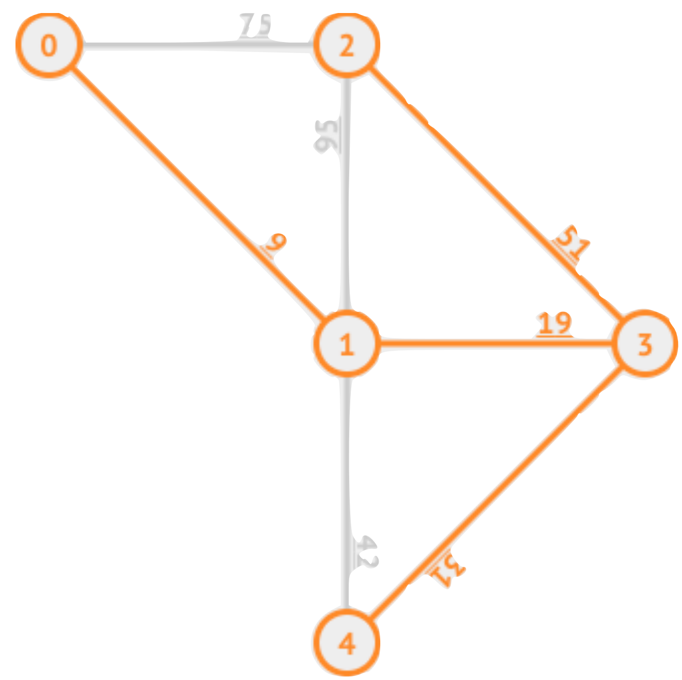
```
traceroute -I -e www.engineering.unsw.edu.au
traceroute to www.engineering.unsw.edu.au (149.171.158.109), 64 hops max, 72 byte packets
 1  csmx-west-rtr.sunet (171.67.64.2)  7.414 ms  9.155 ms  8.288 ms
 2  gnat-2.sunet (172.24.70.12)  0.339 ms  1.532 ms  0.423 ms
 3  csmx-west-rtr-vl3866.sunet (171.64.66.2)  38.916 ms  10.506 ms  8.402 ms
 4  dca-rtr-vlan8.sunet (171.64.255.204)  0.530 ms  0.521 ms  0.713 ms
 5  dc-svl-agg4--stanford-10ge.cenic.net (137.164.50.157)  1.554 ms  1.653 ms  2.828 ms
 6  hpr-svl-hpr2--svl-agg4-10ge.cenic.net (137.164.26.249)  1.212 ms  1.161 ms  1.204 ms
 7  aarnet-2-is-jmb-778.sttlwa.pacificwave.net (207.231.245.4)  17.994 ms  17.998 ms  18.319 ms
 8  et-2-0-0.pe2.brwy.nsw.aarnet.net.au (113.197.15.98)  160.020 ms  160.234 ms  159.922 ms
 9  et-3-3-0.pe1.brwy.nsw.aarnet.net.au (113.197.15.148)  160.285 ms  160.076 ms  160.118 ms
10  138.44.5.1 (138.44.5.1)  160.124 ms  160.138 ms  160.068 ms
11  ombcr1-te-1-5.gw.unsw.edu.au (149.171.255.106)  160.090 ms  160.381 ms  160.185 ms
12  r1dcdnex1-po-2.gw.unsw.edu.au (149.171.255.178)  160.909 ms  160.847 ms  160.921 ms
13  dcfw1-ae-1-3049.gw.unsw.edu.au (129.94.254.60)  160.592 ms  160.558 ms  160.949 ms
14  www.engineering.unsw.edu.au (149.171.158.109)  160.978 ms  161.184 ms  160.987 ms
```

161 milliseconds to get to the final computer

Spanning Trees and Minimum Spanning Trees

Definition: A **Spanning Tree (ST)** of a connected undirected weighted graph **G** is a subgraph of **G** that is a **tree** and **connects (spans) all vertices of G**. A graph **G** can have multiple STs. A **Minimum Spanning Tree (MST)** of **G** is a ST of **G** that has the **smallest total weight** among the various STs. A graph **G** can have multiple MSTs but the MST weight is unique.





Minimum Spanning Tree

Kruskal's Algorithm to find a Minimum Spanning Tree

- **Kruskal's algorithm**: Finds a MST in a given graph.

function **kruskal**(graph):

Remove all edges from the graph.

Place all edges into a **priority queue** based on their weight (cost).

While the priority queue is not empty:

    Dequeue an edge *e* from the priority queue.

    If *e*'s endpoints aren't already connected to one another,

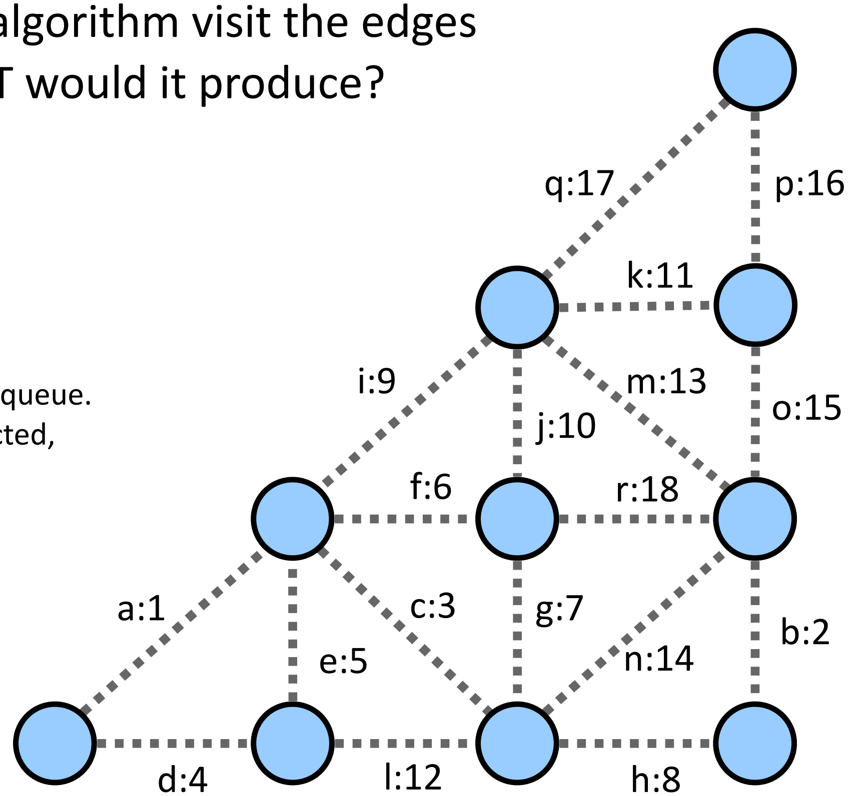
        add that edge into the graph.

    Otherwise, skip the edge.

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
 Remove all edges from the graph.
 Place all edges into a priority queue
  based on their weight (cost).
 While the priority queue is not empty:
  Dequeue an edge *e* from the priority queue.
  If *e*'s endpoints aren't already connected,
   add that edge into the graph.
  Otherwise, skip the edge.

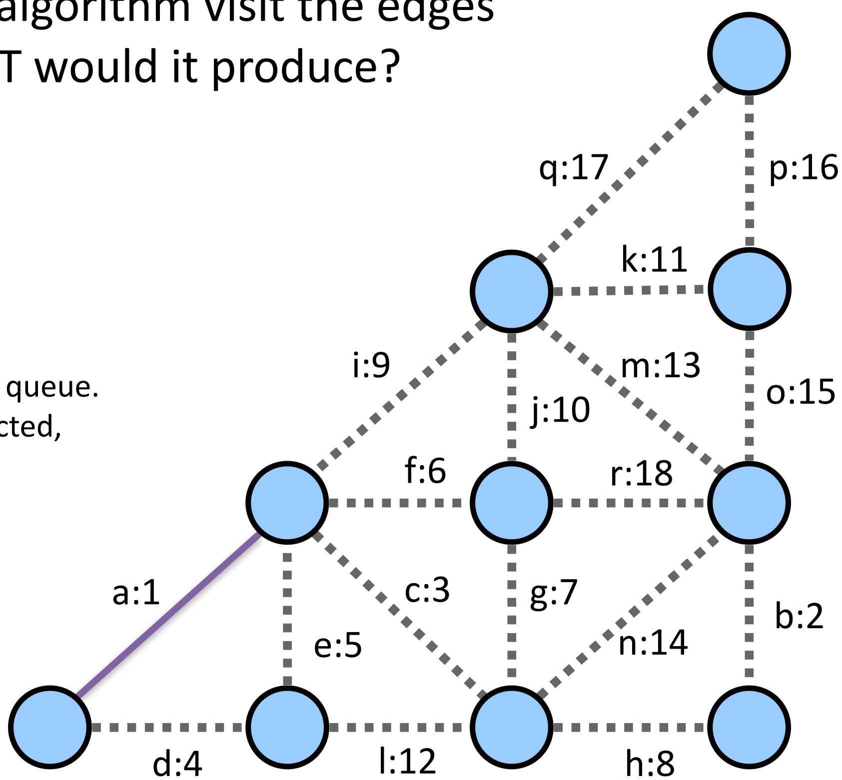


pq = {a:1, b:2, c:3, d:4, e:5, f:6, g:7, h:8, i:9, j:10, k:11, l:12, m:13, n:14, o:15, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
Remove all edges from the graph.
Place all edges into a priority queue based on their weight (cost).
While the priority queue is not empty:
    Dequeue an edge *e* from the priority queue.
    If *e*'s endpoints aren't already connected, add that edge into the graph.
    Otherwise, skip the edge.

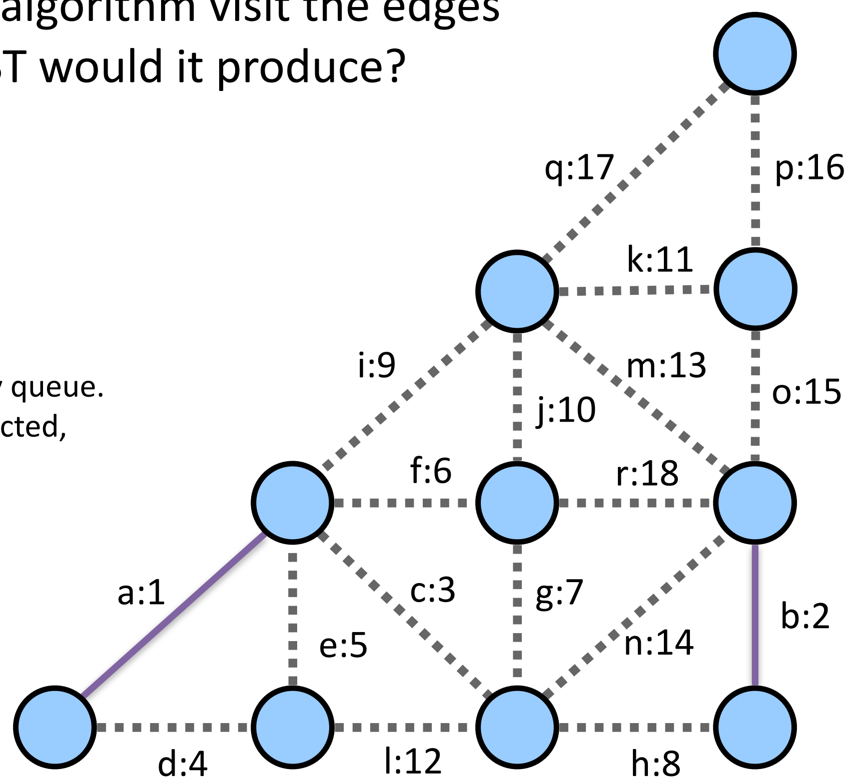


pq = {**a:1**, b:2, c:3, d:4, e:5, f:6, g:7, h:8, i:9, j:10, k:11, l:12, m:13, n:14, o:15, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
 Remove all edges from the graph.
 Place all edges into a priority queue based on their weight (cost).
 While the priority queue is not empty:
  Dequeue an edge *e* from the priority queue.
  If *e*'s endpoints aren't already connected, add that edge into the graph.
  Otherwise, skip the edge.

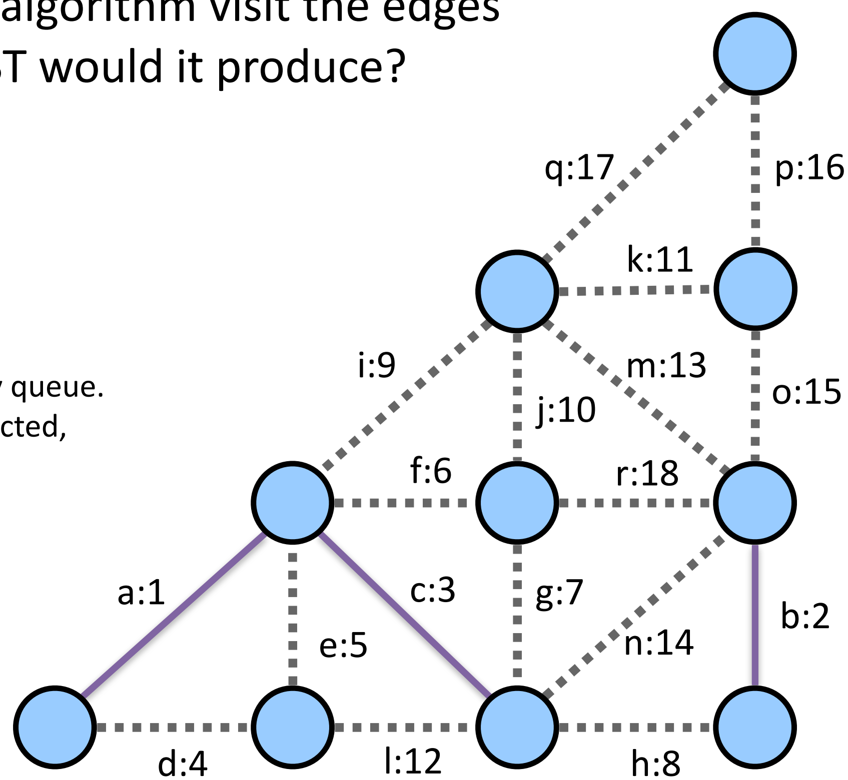


pq = {**b:2**, c:3, d:4, e:5, f:6, g:7, h:8, i:9, j:10, k:11, l:12, m:13, n:14, o:15, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
  Remove all edges from the graph.
  Place all edges into a priority queue
    based on their weight (cost).
  While the priority queue is not empty:
    Dequeue an edge *e* from the priority queue.
    If *e*'s endpoints aren't already connected,
      add that edge into the graph.
    Otherwise, skip the edge.

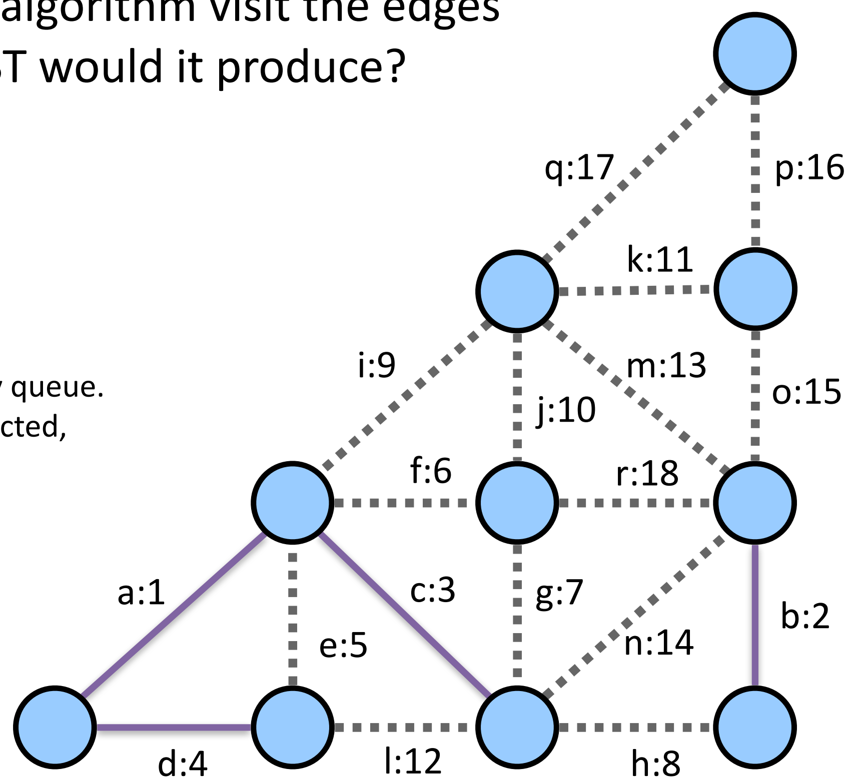


pq = {**c:3**, d:4, e:5, f:6, g:7, h:8, i:9, j:10, k:11, l:12, m:13, n:14, o:15, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
 Remove all edges from the graph.
 Place all edges into a priority queue based on their weight (cost).
 While the priority queue is not empty:
  Dequeue an edge *e* from the priority queue.
  If *e*'s endpoints aren't already connected, add that edge into the graph.
  Otherwise, skip the edge.

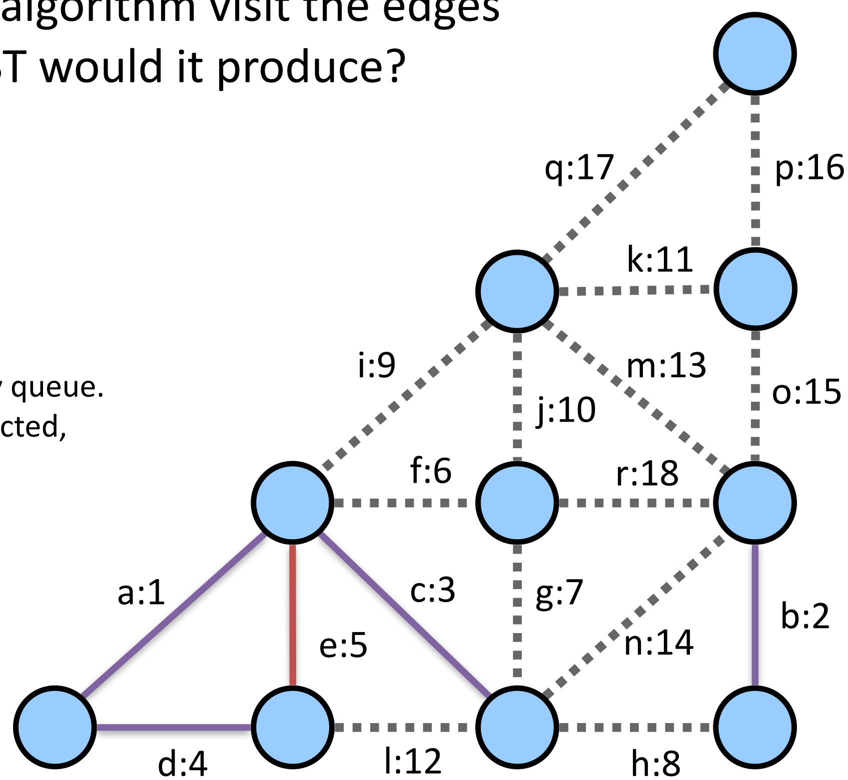


pq = {**d:4**, e:5, f:6, g:7, h:8, i:9, j:10, k:11, l:12, m:13, n:14, o:15, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
 Remove all edges from the graph.
 Place all edges into a priority queue
 based on their weight (cost).
 While the priority queue is not empty:
 Dequeue an edge *e* from the priority queue.
 If *e*'s endpoints aren't already connected,
 add that edge into the graph.
 Otherwise, skip the edge.

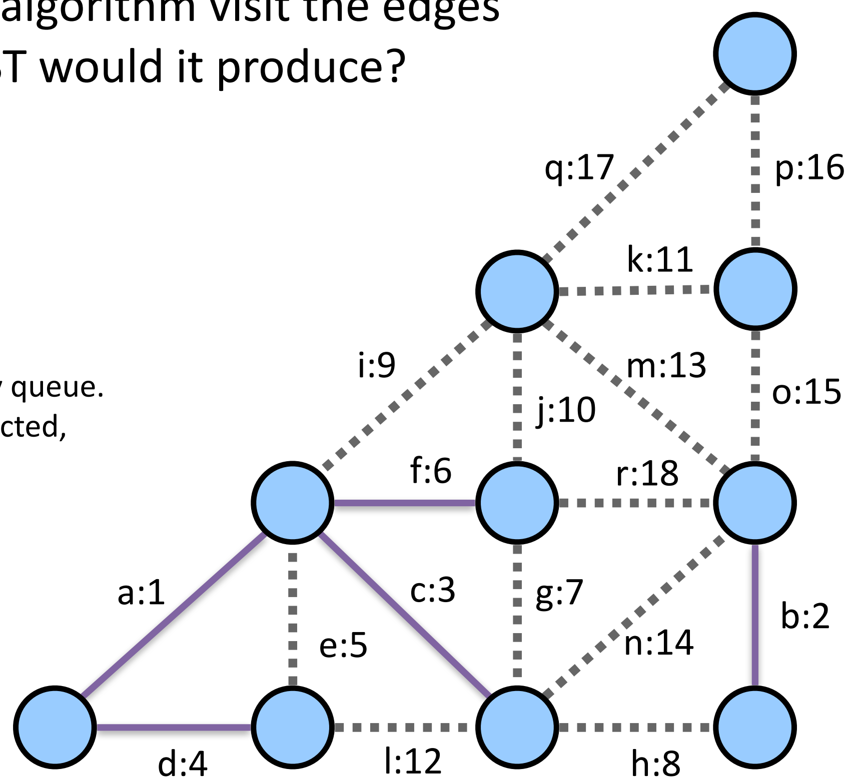


pq = {**e:5**, f:6, g:7, h:8, i:9, j:10, k:11, l:12, m:13, n:14, o:15, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
Remove all edges from the graph.
Place all edges into a priority queue based on their weight (cost).
While the priority queue is not empty:
Dequeue an edge *e* from the priority queue.
If *e*'s endpoints aren't already connected, add that edge into the graph.
Otherwise, skip the edge.

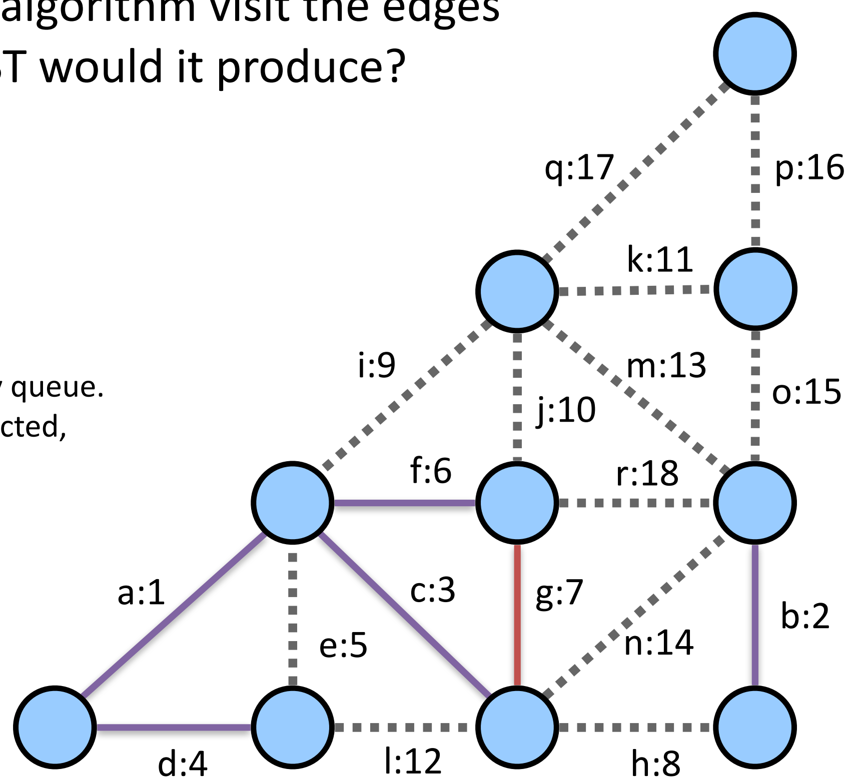


pq = {**f:6**, g:7, h:8, i:9, j:10, k:11, l:12, m:13, n:14, o:15, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
  Remove all edges from the graph.
  Place all edges into a priority queue
    based on their weight (cost).
  While the priority queue is not empty:
    Dequeue an edge *e* from the priority queue.
    If *e*'s endpoints aren't already connected,
      add that edge into the graph.
    Otherwise, skip the edge.

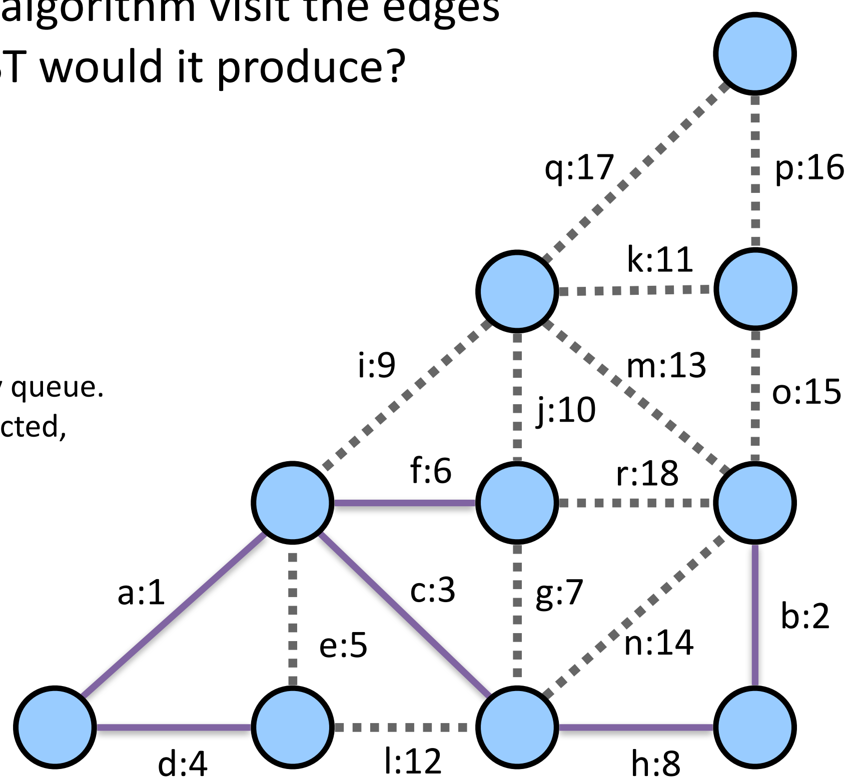


pq = {**g:7**, h:8, i:9, j:10, k:11, l:12, m:13, n:14, o:15, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
 Remove all edges from the graph.
 Place all edges into a priority queue based on their weight (cost).
 While the priority queue is not empty:
  Dequeue an edge *e* from the priority queue.
  If *e*'s endpoints aren't already connected, add that edge into the graph.
  Otherwise, skip the edge.

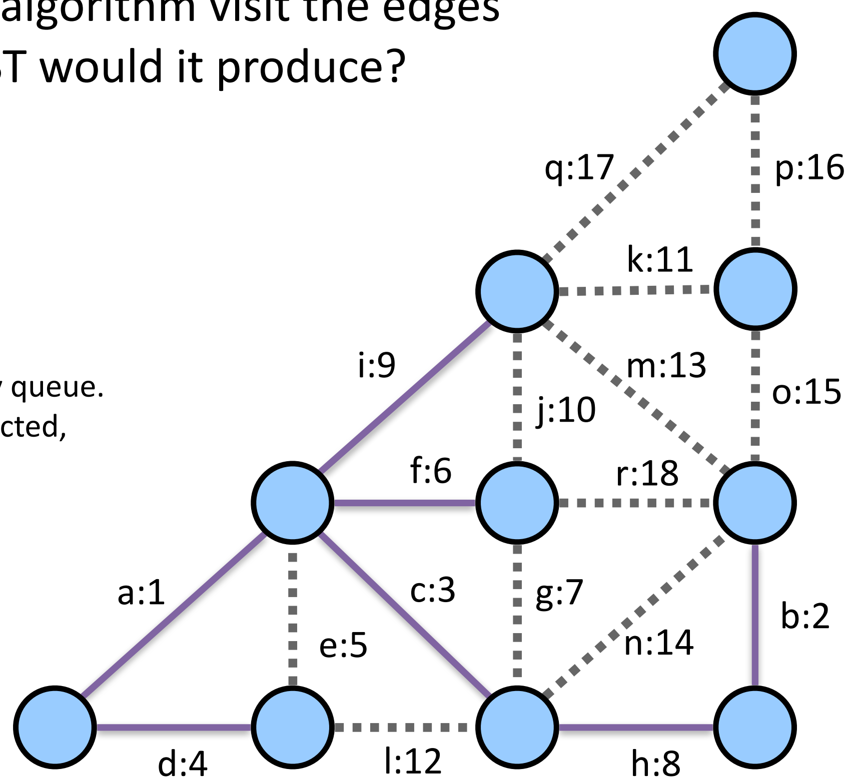


pq = {**h:8**, i:9, j:10, k:11, l:12, m:13, n:14, o:15, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
 Remove all edges from the graph.
 Place all edges into a priority queue
  based on their weight (cost).
 While the priority queue is not empty:
  Dequeue an edge *e* from the priority queue.
  If *e*'s endpoints aren't already connected,
   add that edge into the graph.
  Otherwise, skip the edge.

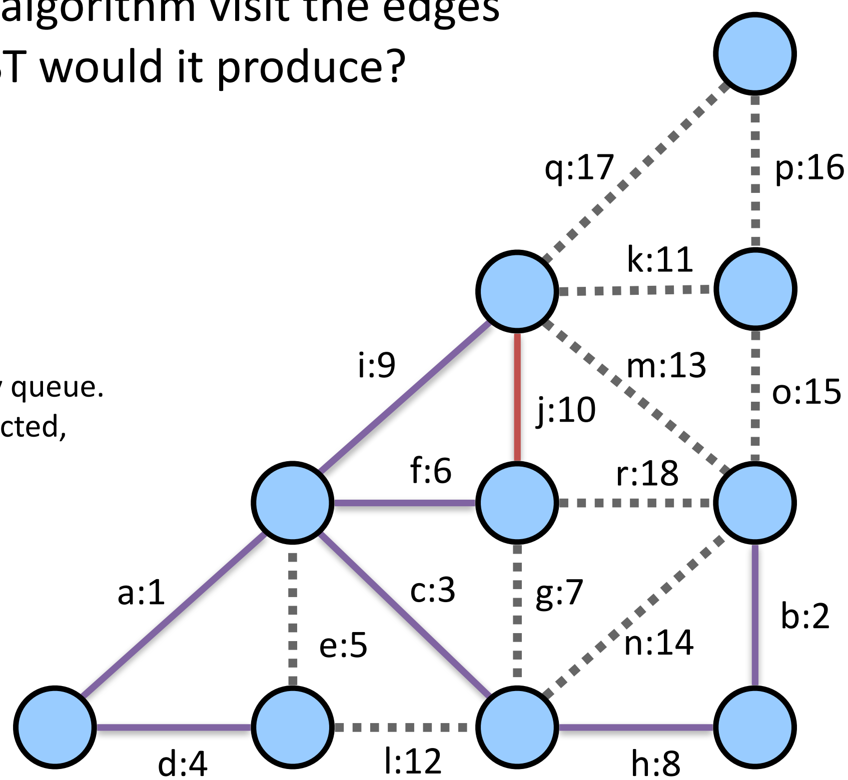


pq = {**i:9**, j:10, k:11, l:12, m:13, n:14, o:15, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
  Remove all edges from the graph.
  Place all edges into a priority queue based on their weight (cost).
  While the priority queue is not empty:
    Dequeue an edge *e* from the priority queue.
    If *e*'s endpoints aren't already connected, add that edge into the graph.
    Otherwise, skip the edge.

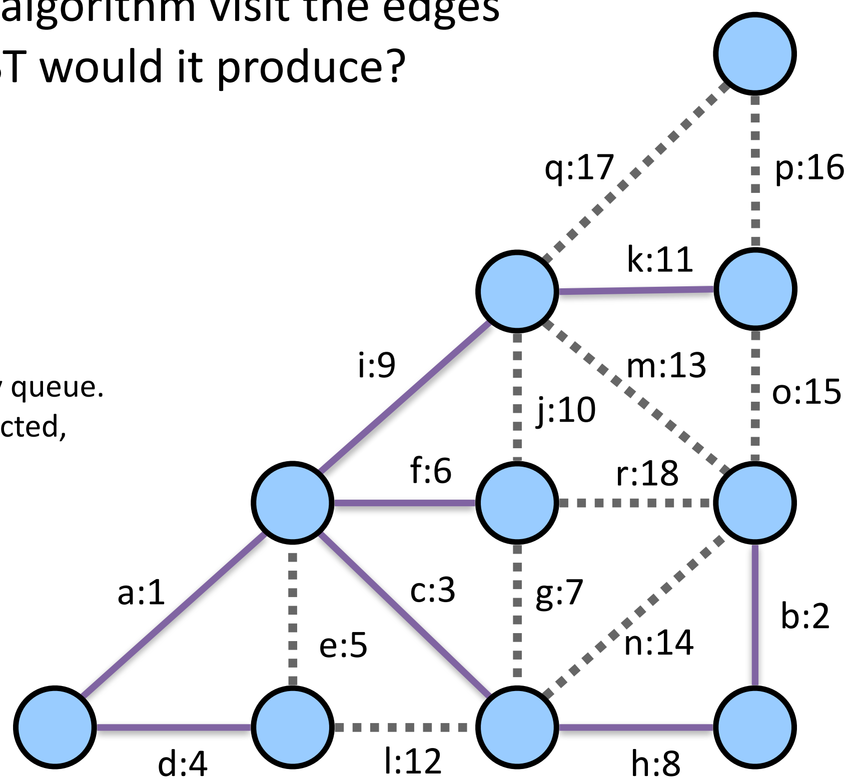


pq = {**j:10**, k:11, l:12, m:13, n:14, o:15, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
 Remove all edges from the graph.
 Place all edges into a priority queue based on their weight (cost).
 While the priority queue is not empty:
  Dequeue an edge *e* from the priority queue.
  If *e*'s endpoints aren't already connected, add that edge into the graph.
  Otherwise, skip the edge.

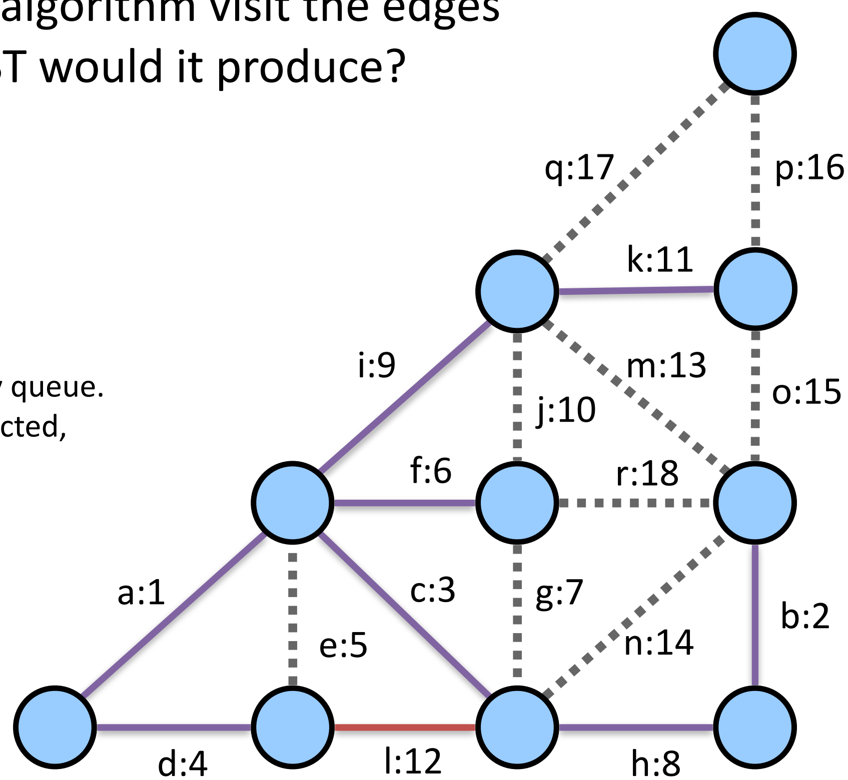


pq = {**k:11**, l:12, m:13, n:14, o:15, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
 Remove all edges from the graph.
 Place all edges into a priority queue
  based on their weight (cost).
 While the priority queue is not empty:
  Dequeue an edge *e* from the priority queue.
  If *e*'s endpoints aren't already connected,
   add that edge into the graph.
  Otherwise, skip the edge.

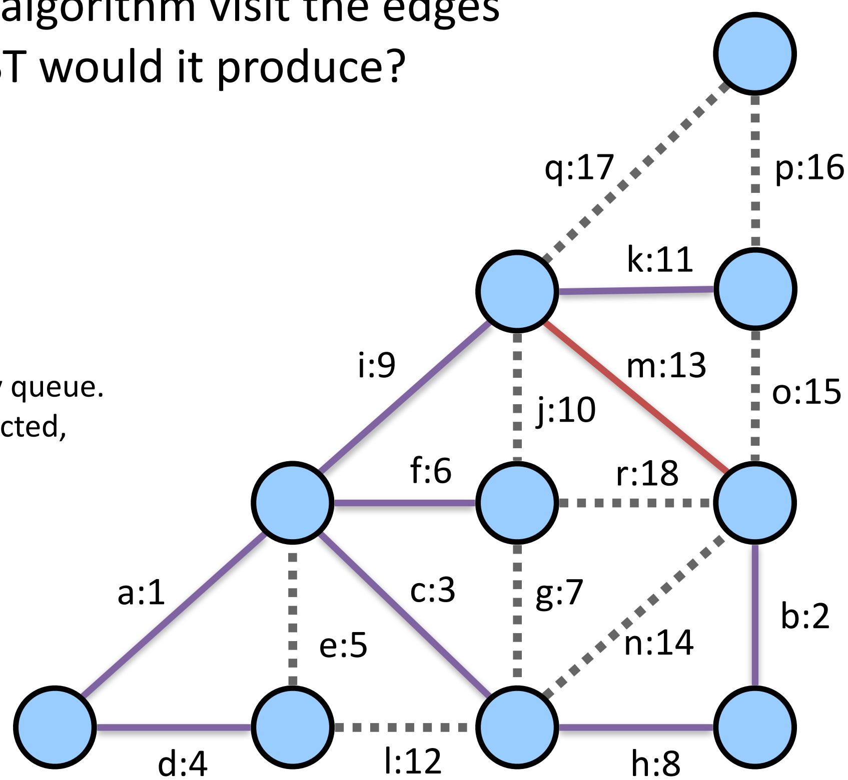


pq = {**l:12**, m:13, n:14, o:15, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
Remove all edges from the graph.
Place all edges into a priority queue based on their weight (cost).
While the priority queue is not empty:
  Dequeue an edge *e* from the priority queue.
  If *e*'s endpoints aren't already connected, add that edge into the graph.
  Otherwise, skip the edge.

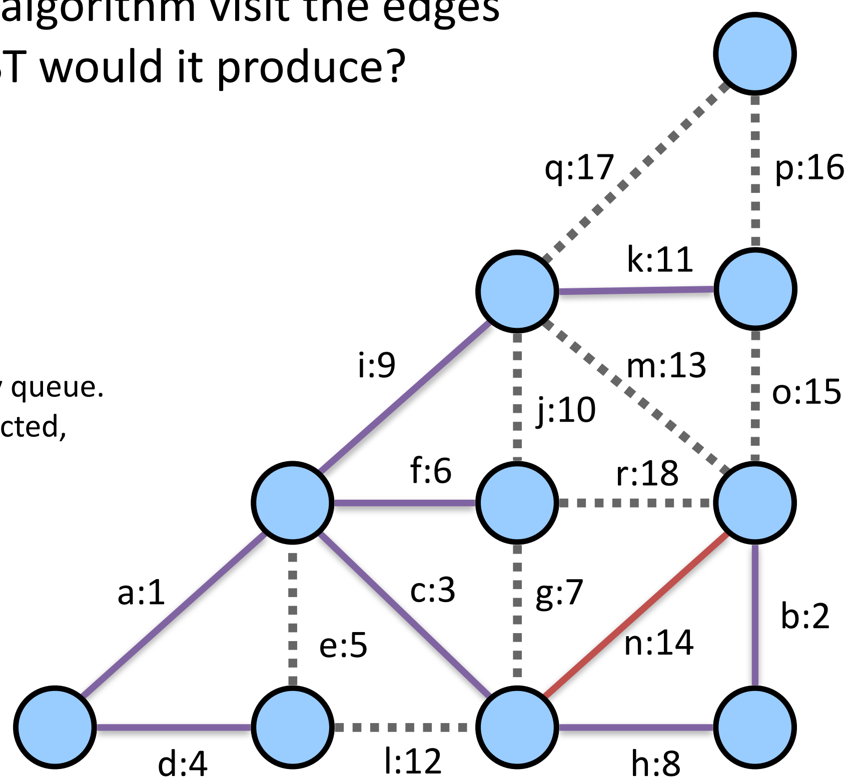


pq = {**m:13**, n:14, o:15, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
Remove all edges from the graph.
Place all edges into a priority queue based on their weight (cost).
While the priority queue is not empty:
Dequeue an edge *e* from the priority queue.
If *e*'s endpoints aren't already connected, add that edge into the graph.
Otherwise, skip the edge.

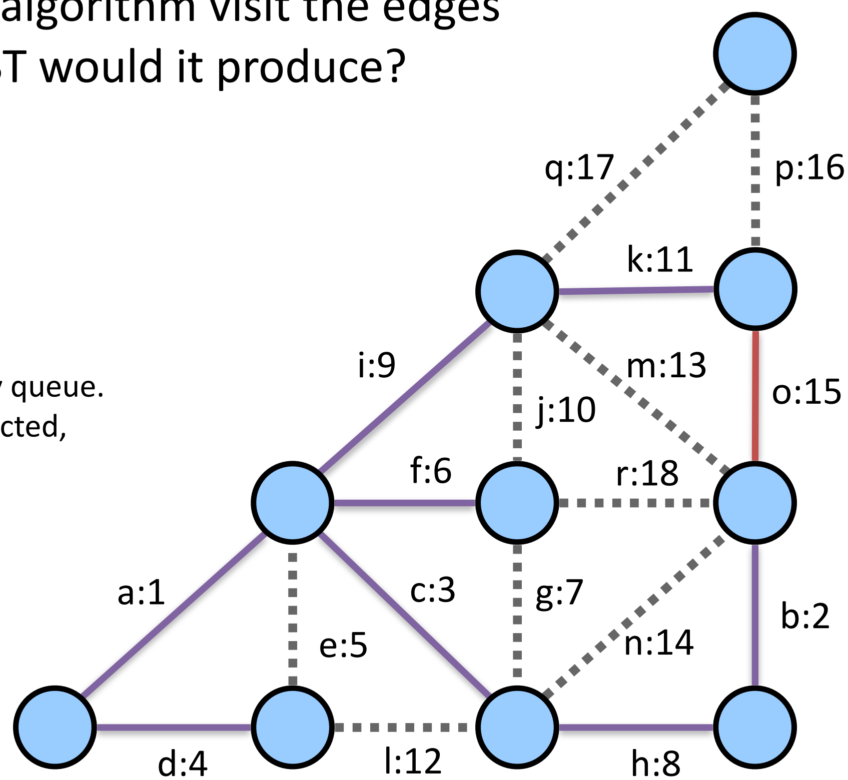


pq = {**n:14**, o:15, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
Remove all edges from the graph.
Place all edges into a priority queue based on their weight (cost).
While the priority queue is not empty:
Dequeue an edge *e* from the priority queue.
If *e*'s endpoints aren't already connected, add that edge into the graph.
Otherwise, skip the edge.

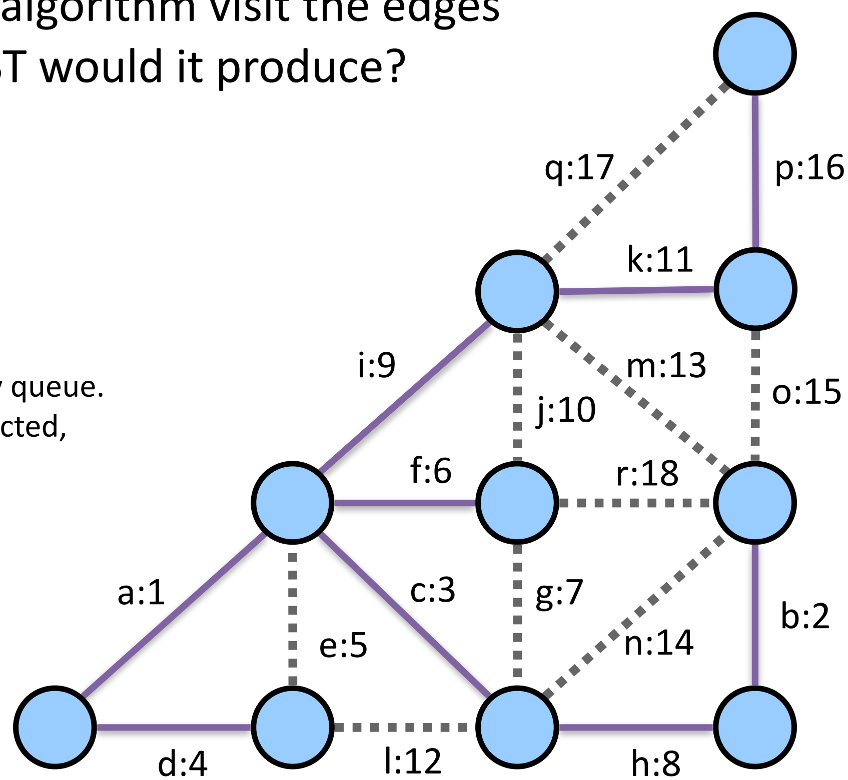


pq = {**o:15**, p:16, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
 Remove all edges from the graph.
 Place all edges into a priority queue based on their weight (cost).
 While the priority queue is not empty:
  Dequeue an edge *e* from the priority queue.
  If *e*'s endpoints aren't already connected, add that edge into the graph.
  Otherwise, skip the edge.

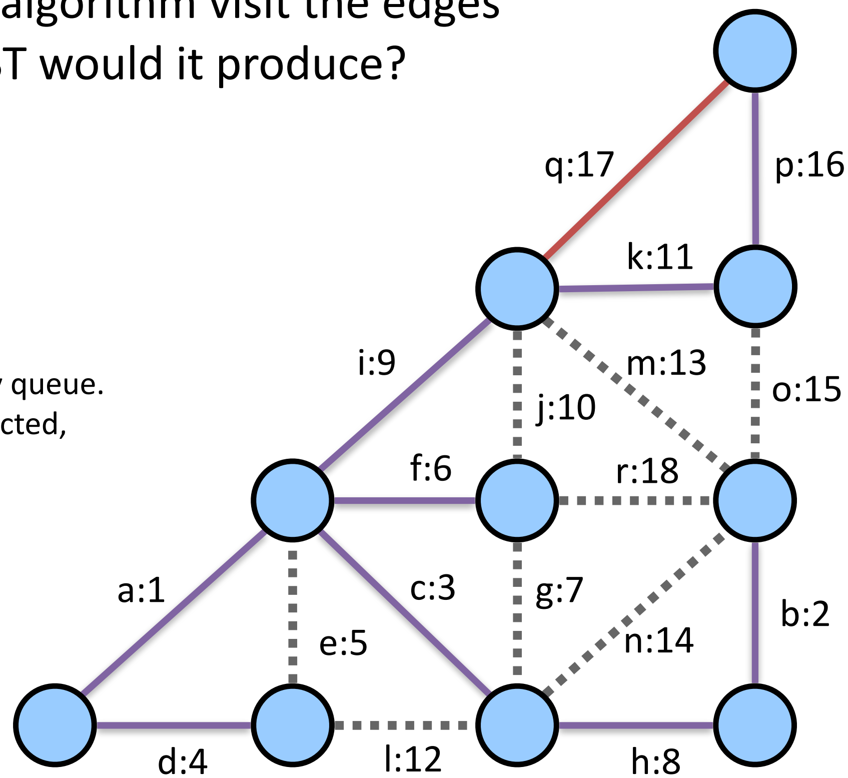


pq = {**p:16**, q:17, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
 Remove all edges from the graph.
 Place all edges into a priority queue based on their weight (cost).
 While the priority queue is not empty:
  Dequeue an edge *e* from the priority queue.
  If *e*'s endpoints aren't already connected, add that edge into the graph.
  Otherwise, skip the edge.

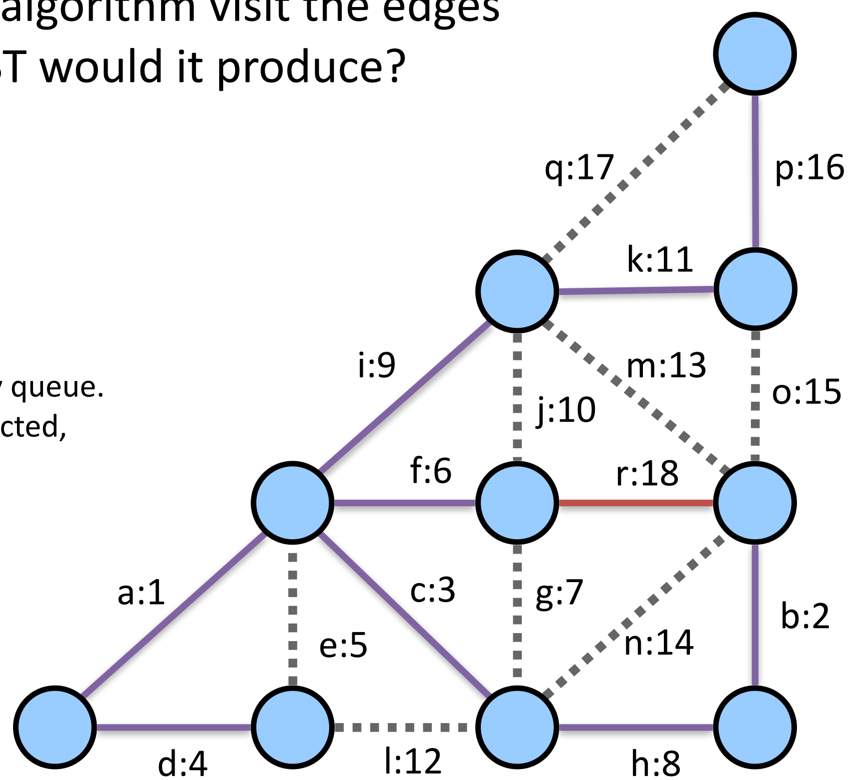


pq = {**q:17**, r:18}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
 Remove all edges from the graph.
 Place all edges into a priority queue based on their weight (cost).
 While the priority queue is not empty:
  Dequeue an edge *e* from the priority queue.
  If *e*'s endpoints aren't already connected, add that edge into the graph.
  Otherwise, skip the edge.

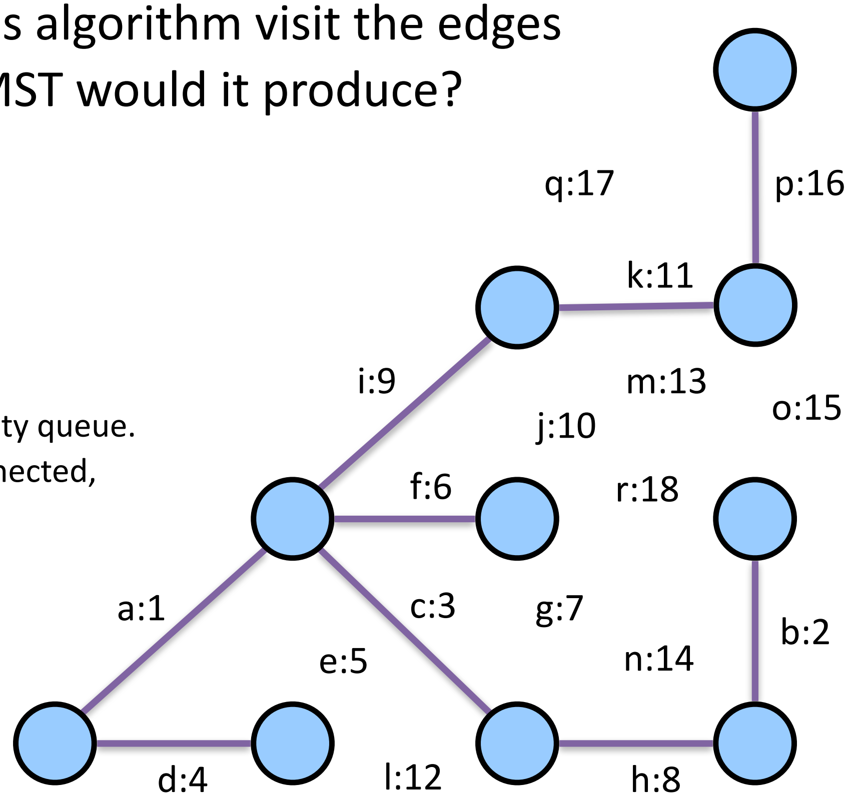


pq = {**r:18**}

Kruskal Example

- In what order would Kruskal's algorithm visit the edges in the graph below? What MST would it produce?

function **kruskal**(graph):
  Remove all edges from the graph.
  Place all edges into a priority queue based on their weight (cost).
  While the priority queue is not empty:
    Dequeue an edge *e* from the priority queue.
    If *e*'s endpoints aren't already connected, add that edge into the graph.
    Otherwise, skip the edge.



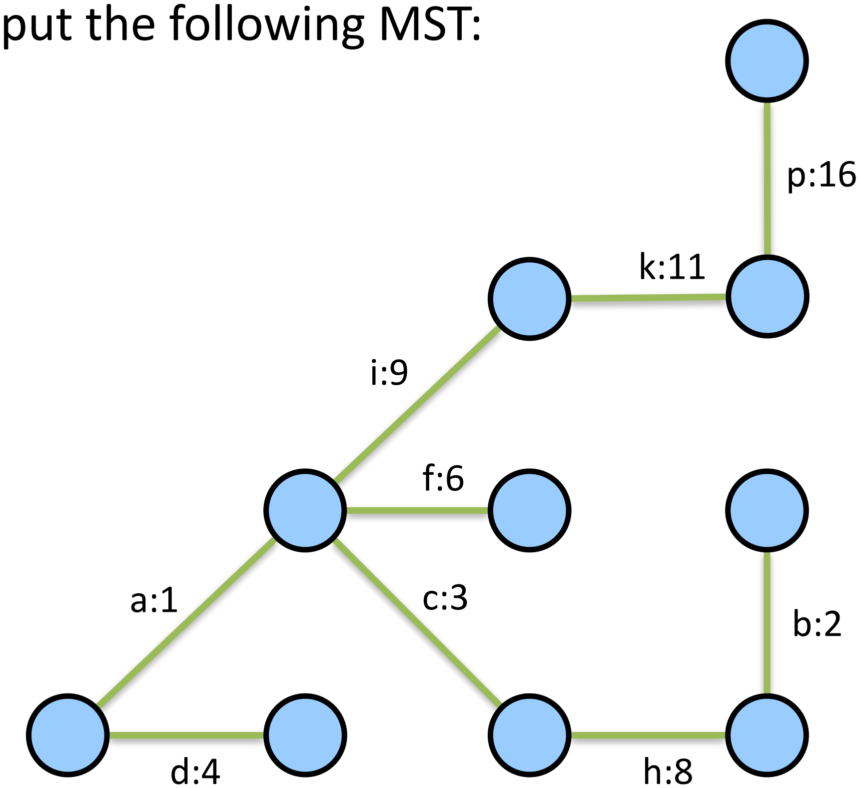
pq = {}

Kruskal Example

- Kruskal's algorithm would output the following MST:
  - {a, b, c, d, f, h, i, k, p}

- The MST's total cost is:

  1+2+3+4+6+8+9+11+16 = 60



- What data structures should we use to implement this algorithm?

function **kruskal**(graph):

Remove all edges from the graph.

Place all edges into a **<u>priority queue</u>**
based on their weight (cost).

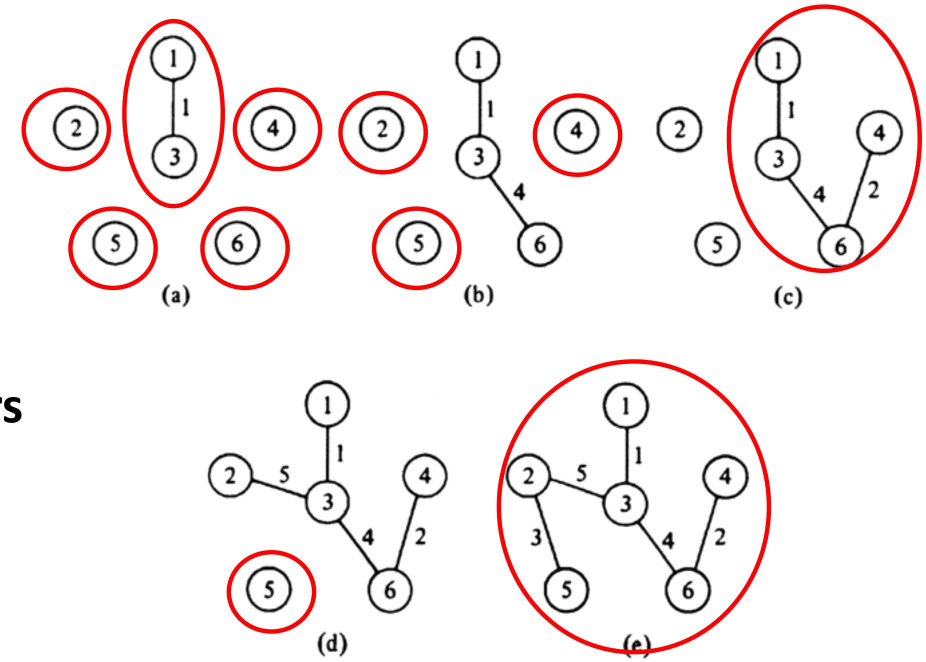
While the priority queue is not empty:

Dequeue an edge *e* from the priority queue.

**If *e*'s endpoints aren't already connected,
add that edge into the graph.**

Otherwise, skip the edge.

- Need some way to identify which vertexes are "connected" to which other ones
  - we call these "**clusters**" of vertices
- Also need an efficient way to figure out which cluster a given vertex is in.
- Also need to **merge clusters** when adding an edge.



References and Advanced Reading

- **References:**
  - Minimum Spanning Tree visualization: https://visualgo.net/mst
  - Kruskal's Algorithm: https://en.wikipedia.org/wiki/Kruskal's_algorithm

- **Advanced Reading:**
  - How Internet Routing works: https://web.stanford.edu/class/msande91si/www-spr04/readings/week1/InternetWhitepaper.htm
  - http://www.explainthatstuff.com/internet.html

Extra Slides