CSCI 232:
Data Structures and Algorithms

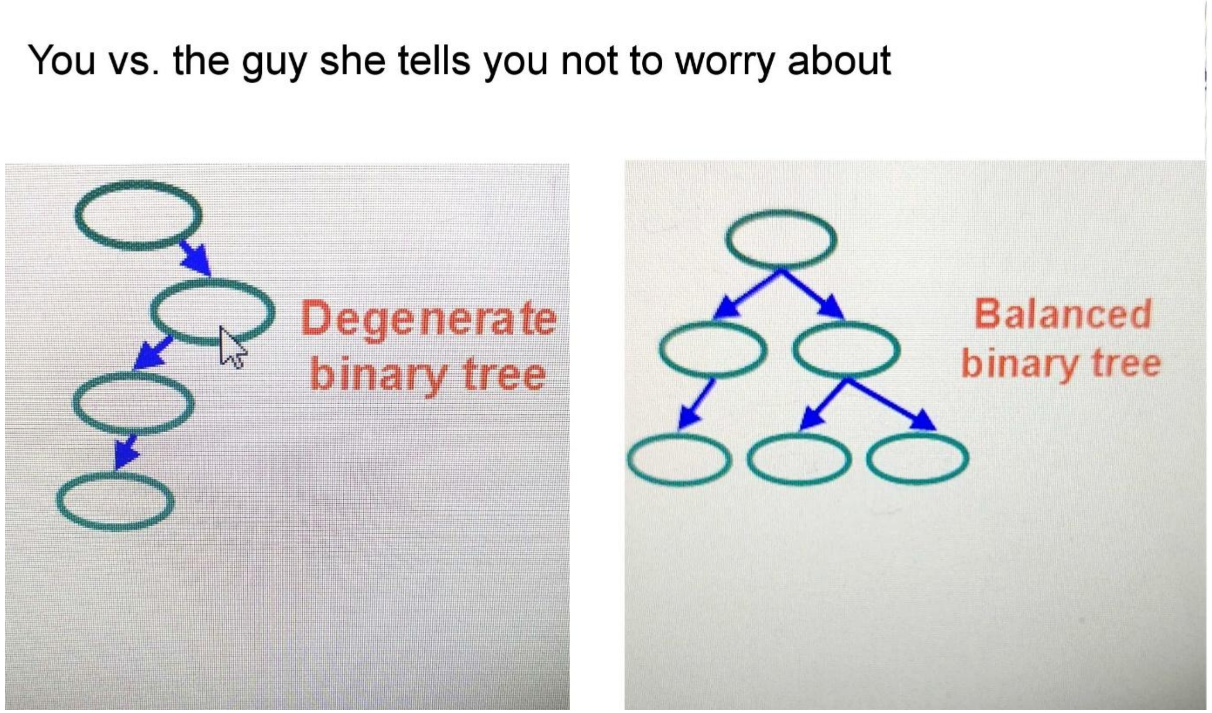
Graphs (Representation)

Reese Pearsall

Spring 2025

Announcements

Lab 7 due on Friday
→ Really easy



Registration

Next Classes (You can register for these anytime in the next couple years):

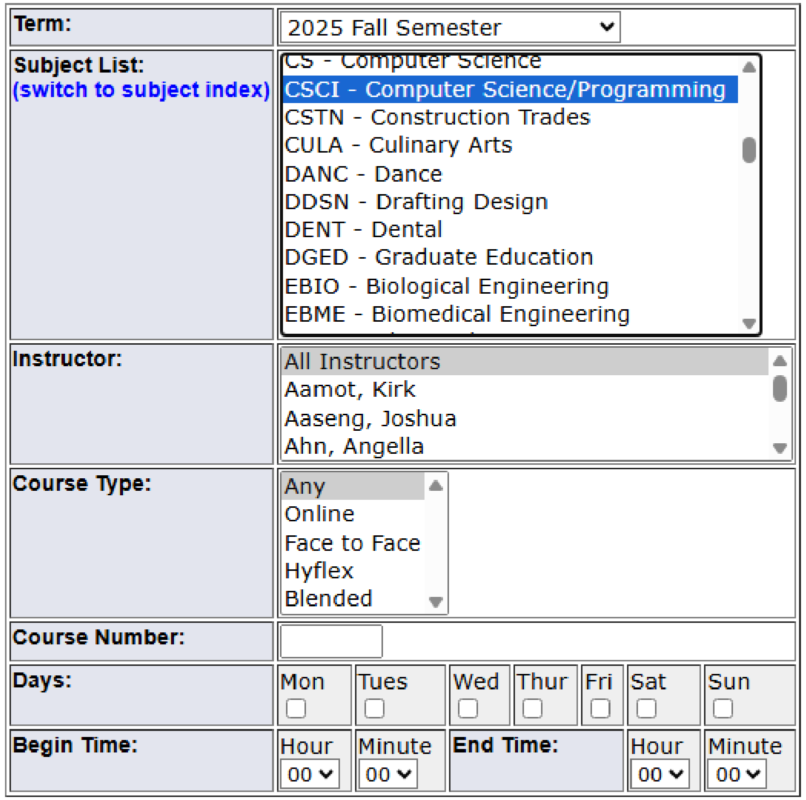
CSCI 366- Computer Systems
ESOF 322 – Software Engineering
CSCI 305 – Concepts of Programming Languages
CSCI 338 – Computer Science Theory

If you have not already:

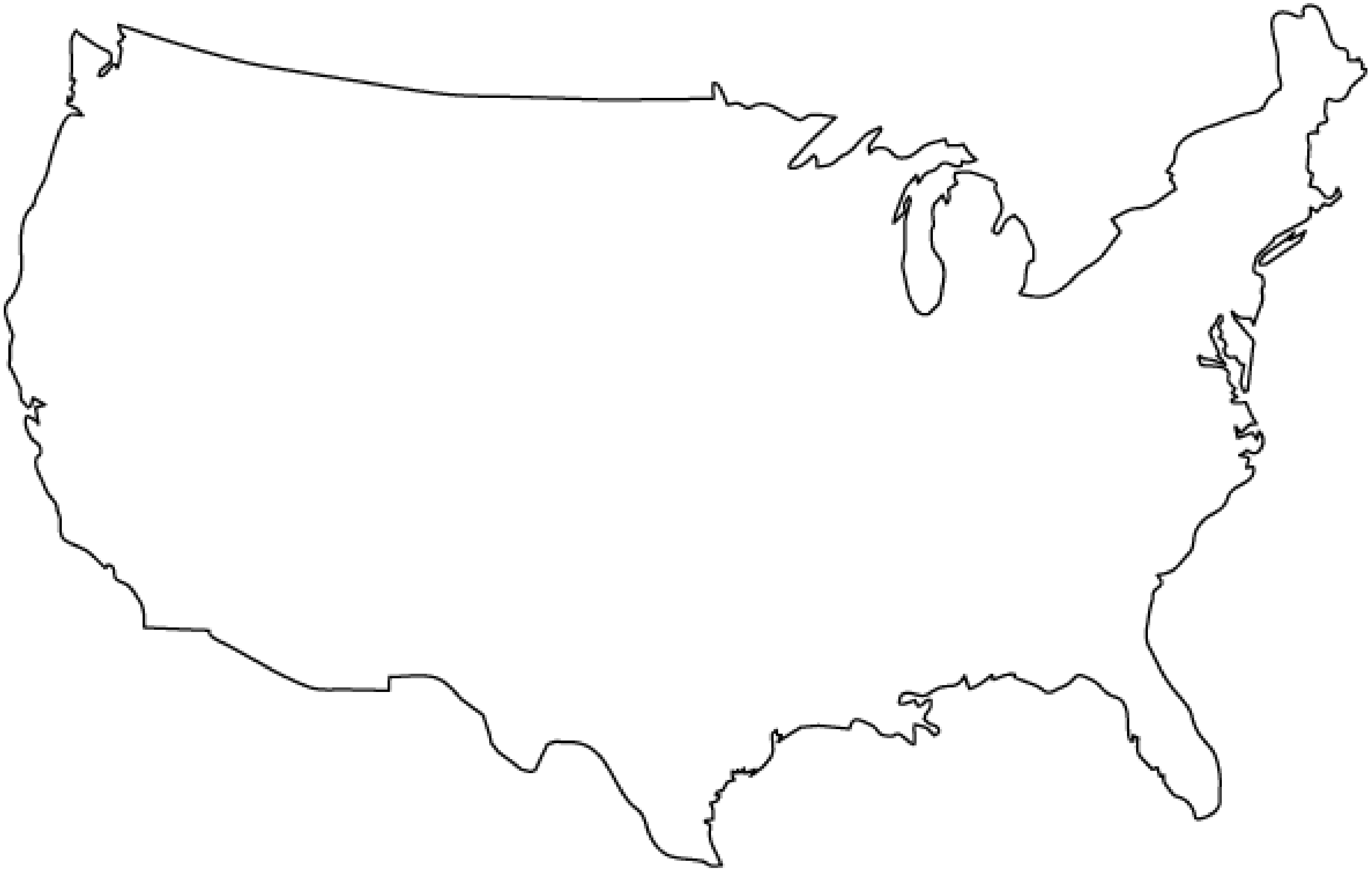
CSCI 246 – Discrete Structures
CSCI 112- Programming in C
MART 145- Web Design

Other Classes that may be of interest

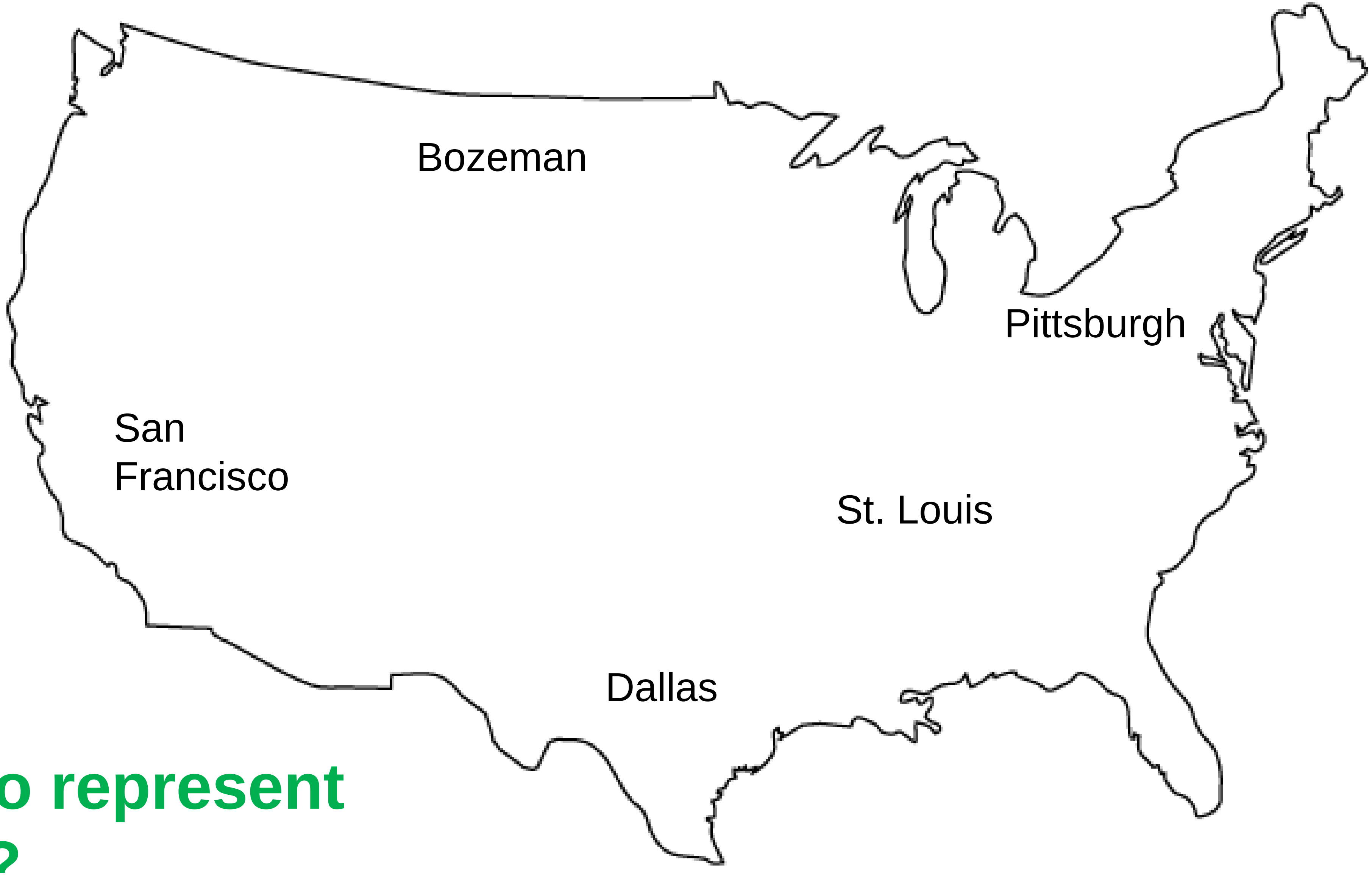
CSCI 252- Intro to Data Science
CSCI 331- Web Development
CSCI 351 – System Administration
CSCI 440 – Database Systems
CSCI 443 – User Interface Design
CSCI 451 – Computational Biology
CSCI 446 – Artificial Intelligence*
CSCI 460 – Operating Systems
CSCI 466 – Networks
CSCI 476 – Computer Security



How could we visualize: The US Road Network?

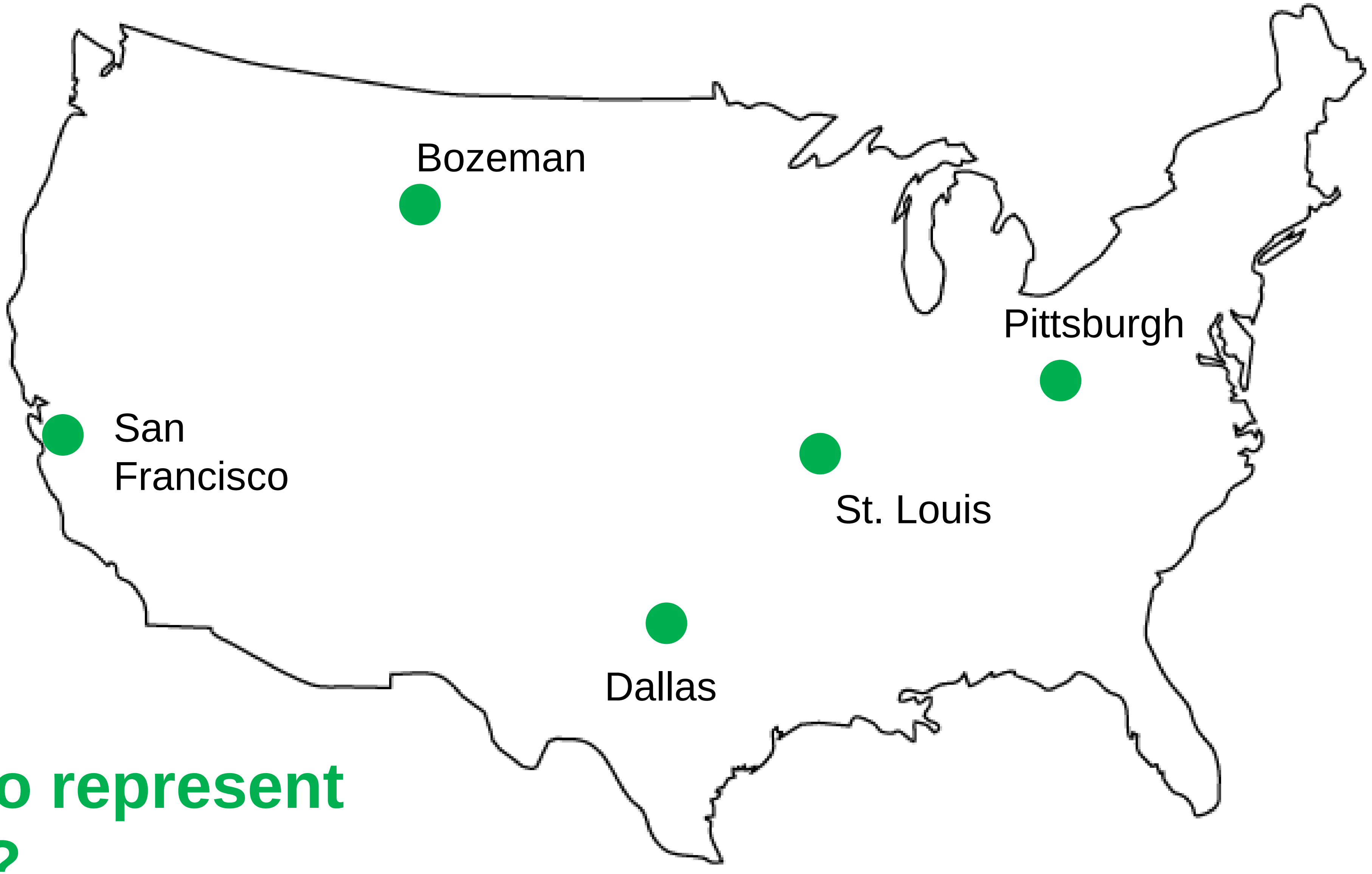


How could we visualize: The US Road Network?



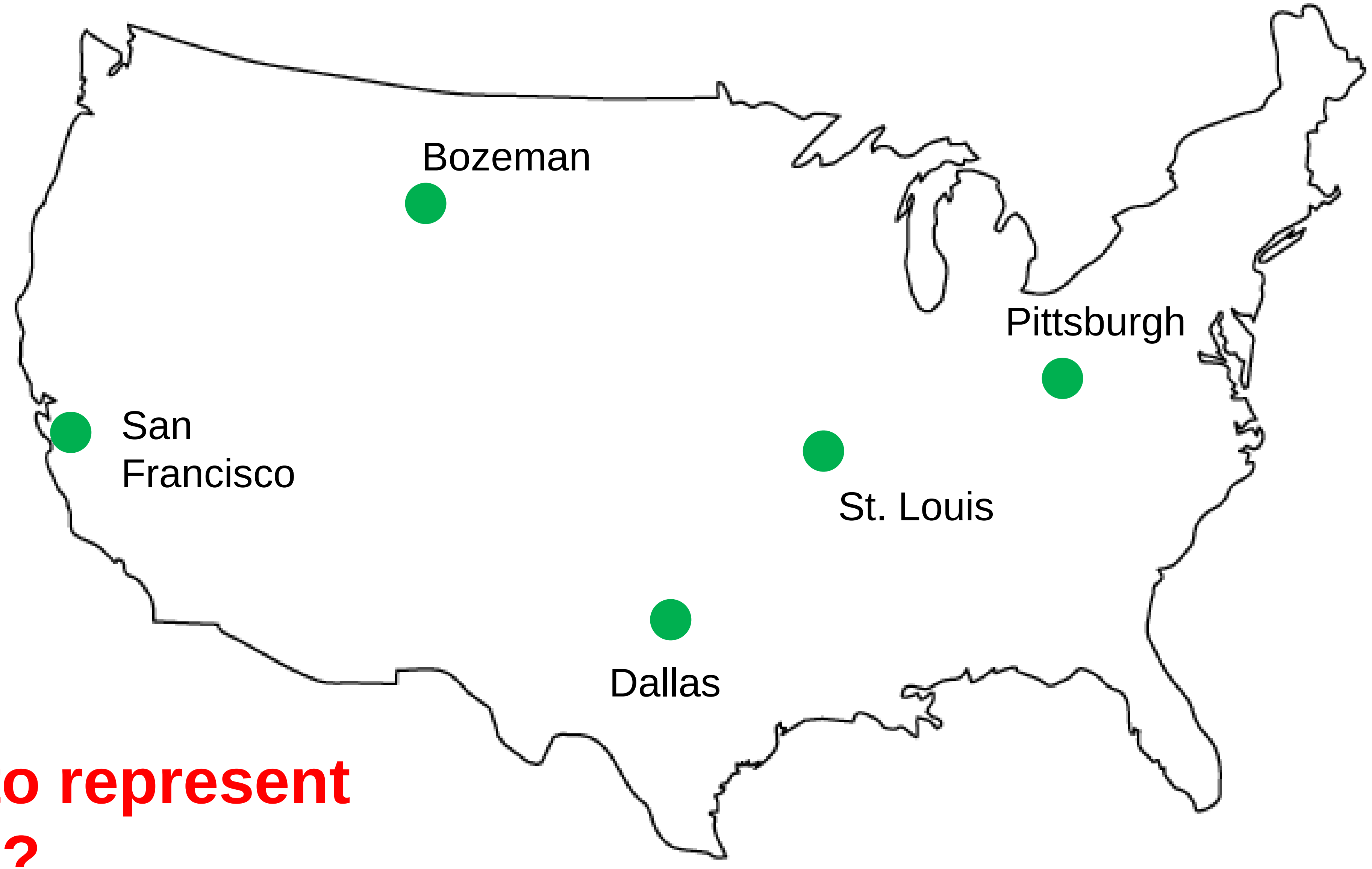
How to represent cities?

How could we visualize: The US Road Network?



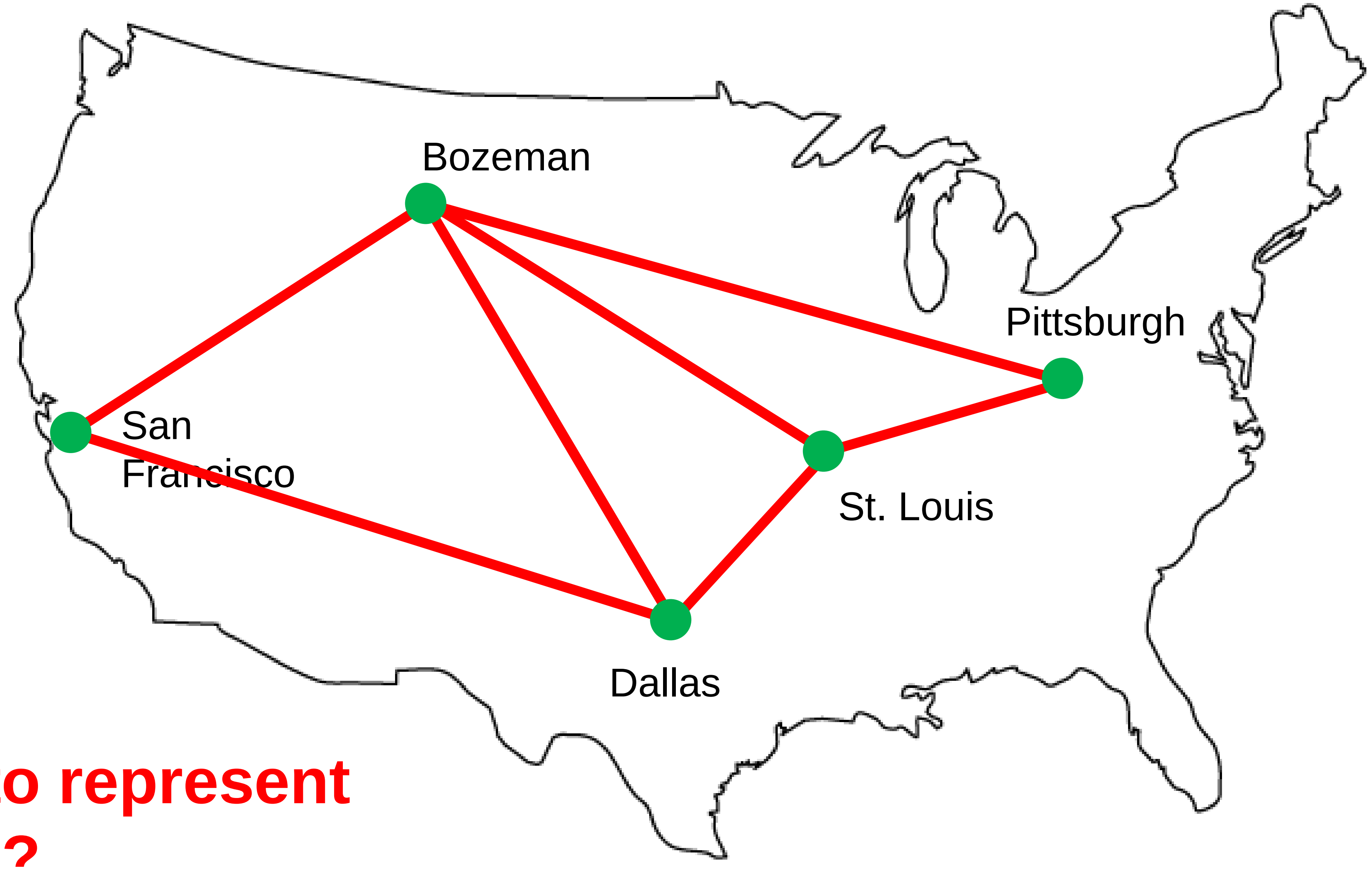
How to represent cities?

How could we visualize: The US Road Network?



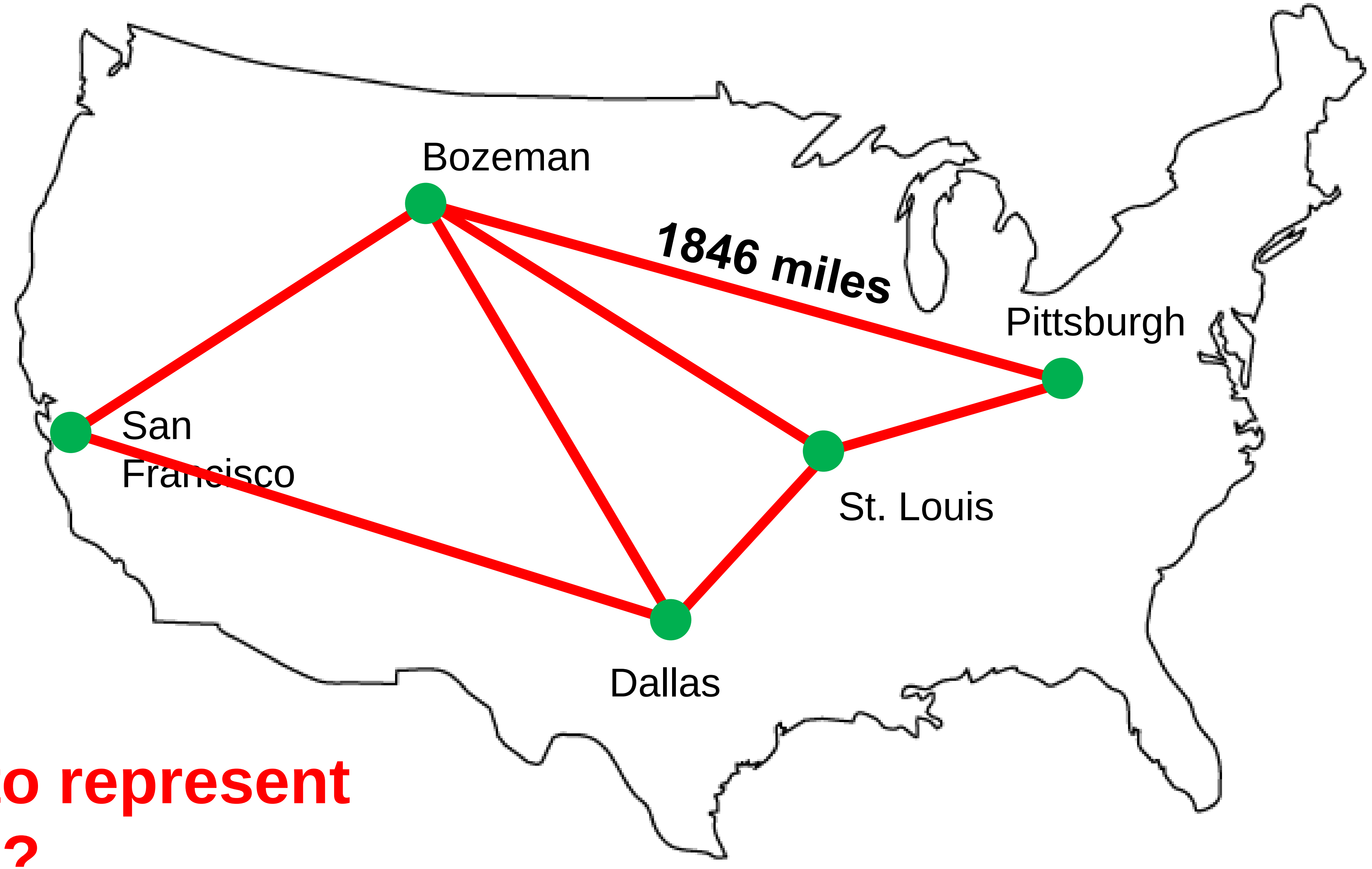
How to represent roads?

How could we visualize: The US Road Network?



How to represent roads?

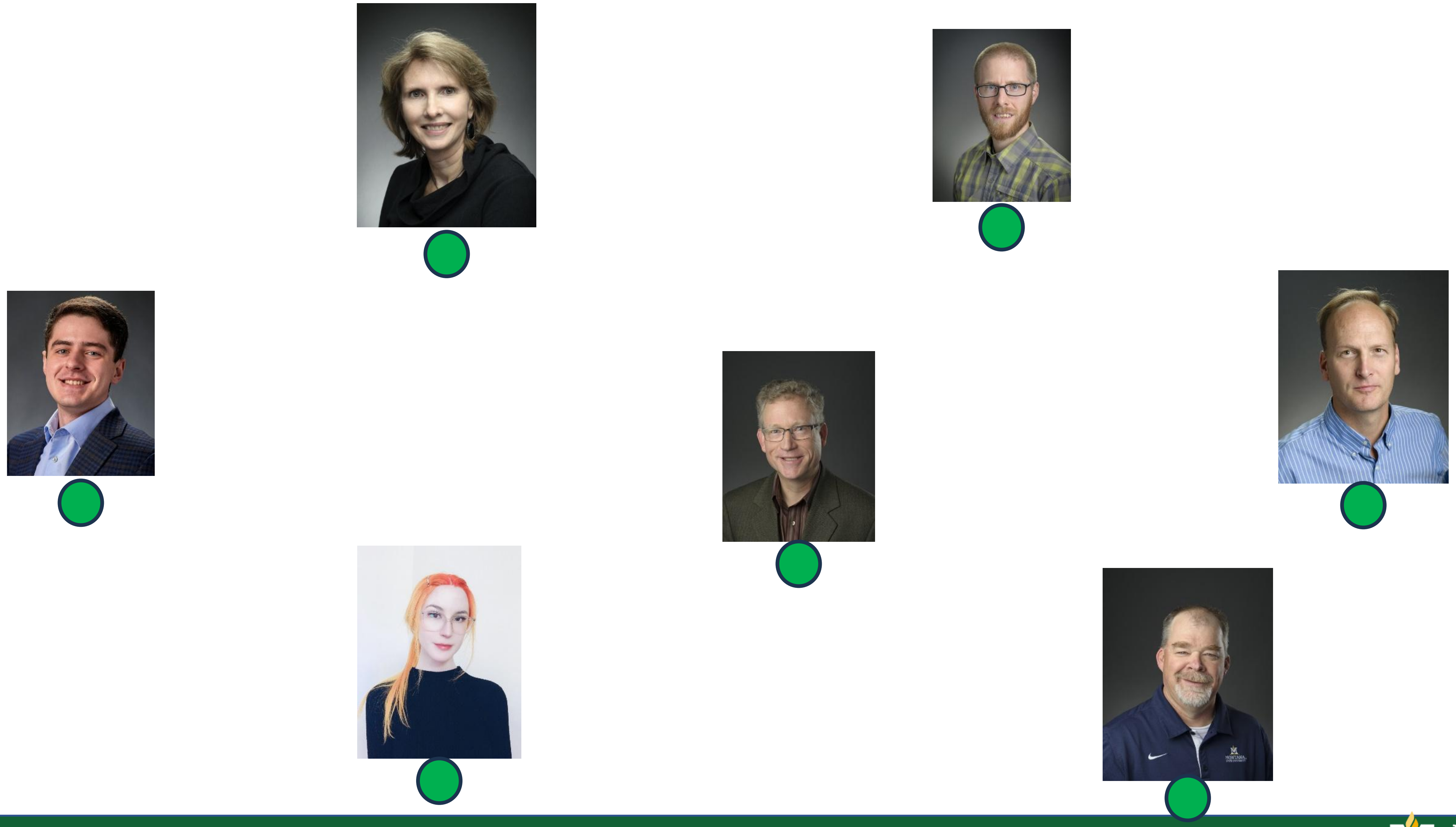
How could we visualize: The US Road Network?



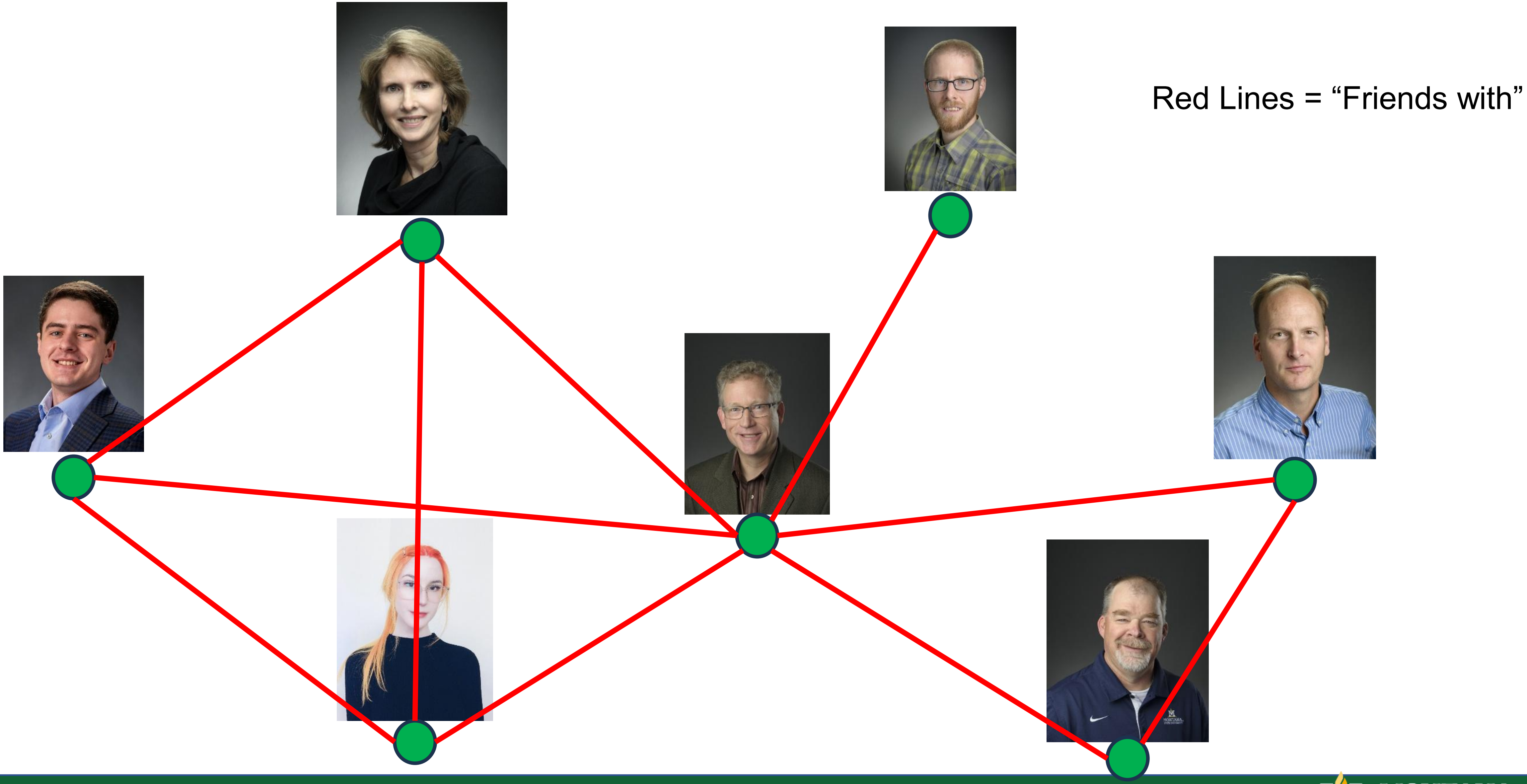
How to represent roads?

How could we visualize: Connections in a Social Media Network?

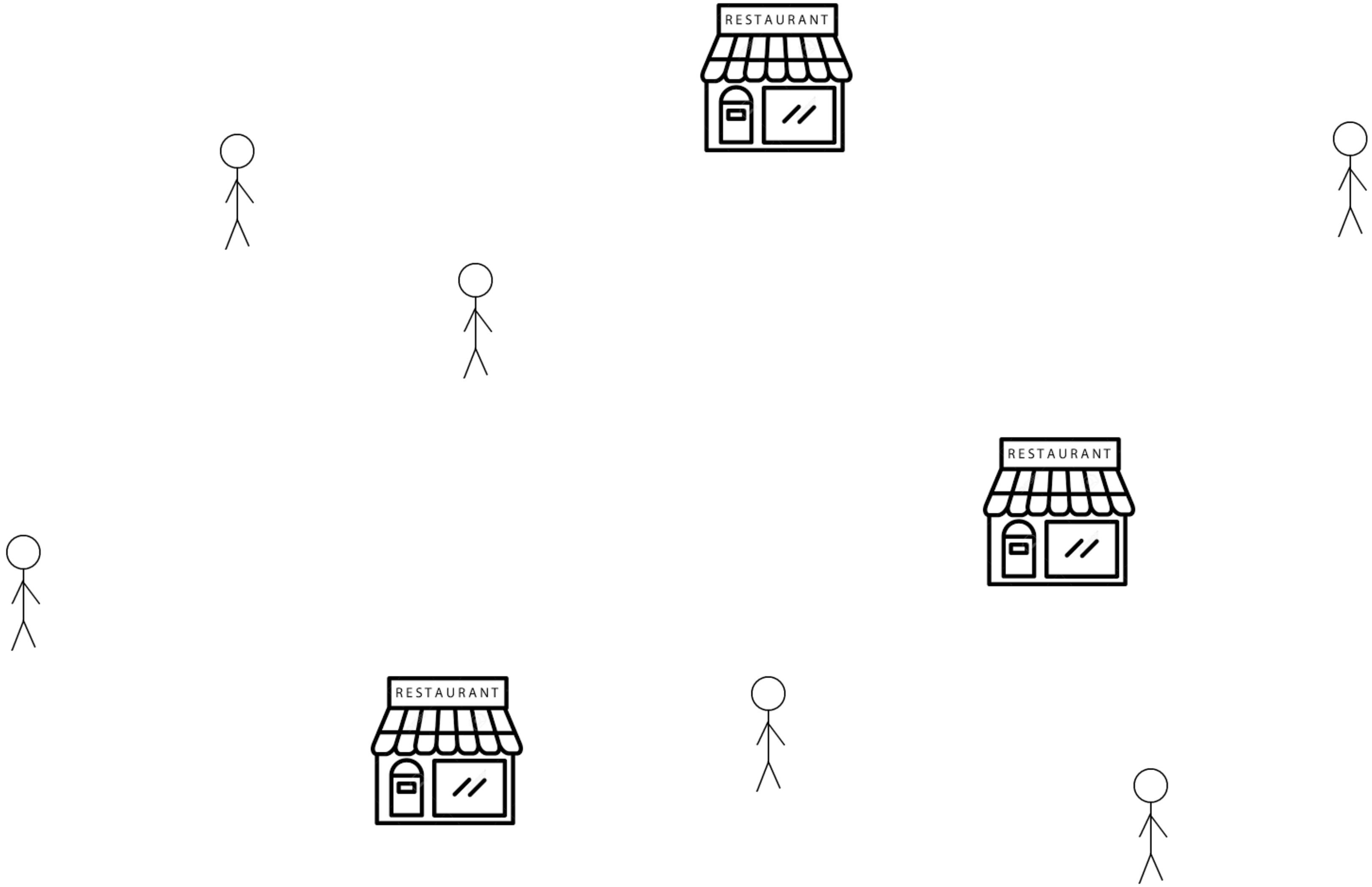
How could we visualize: Connections in a Social Media Network?



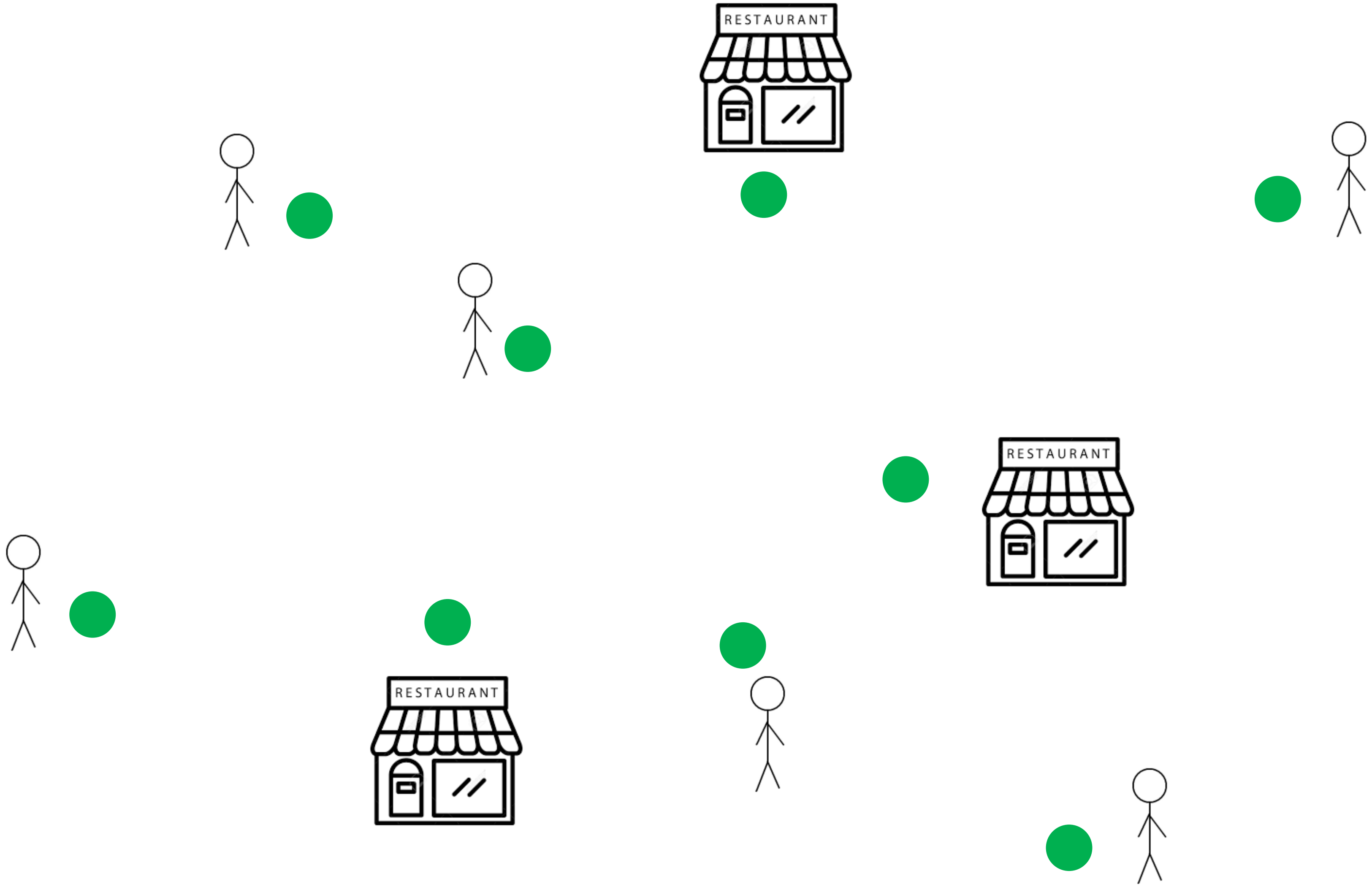
How could we visualize: Connections in a Social Media Network?



How could we visualize: Restaurants and Potential Customers?

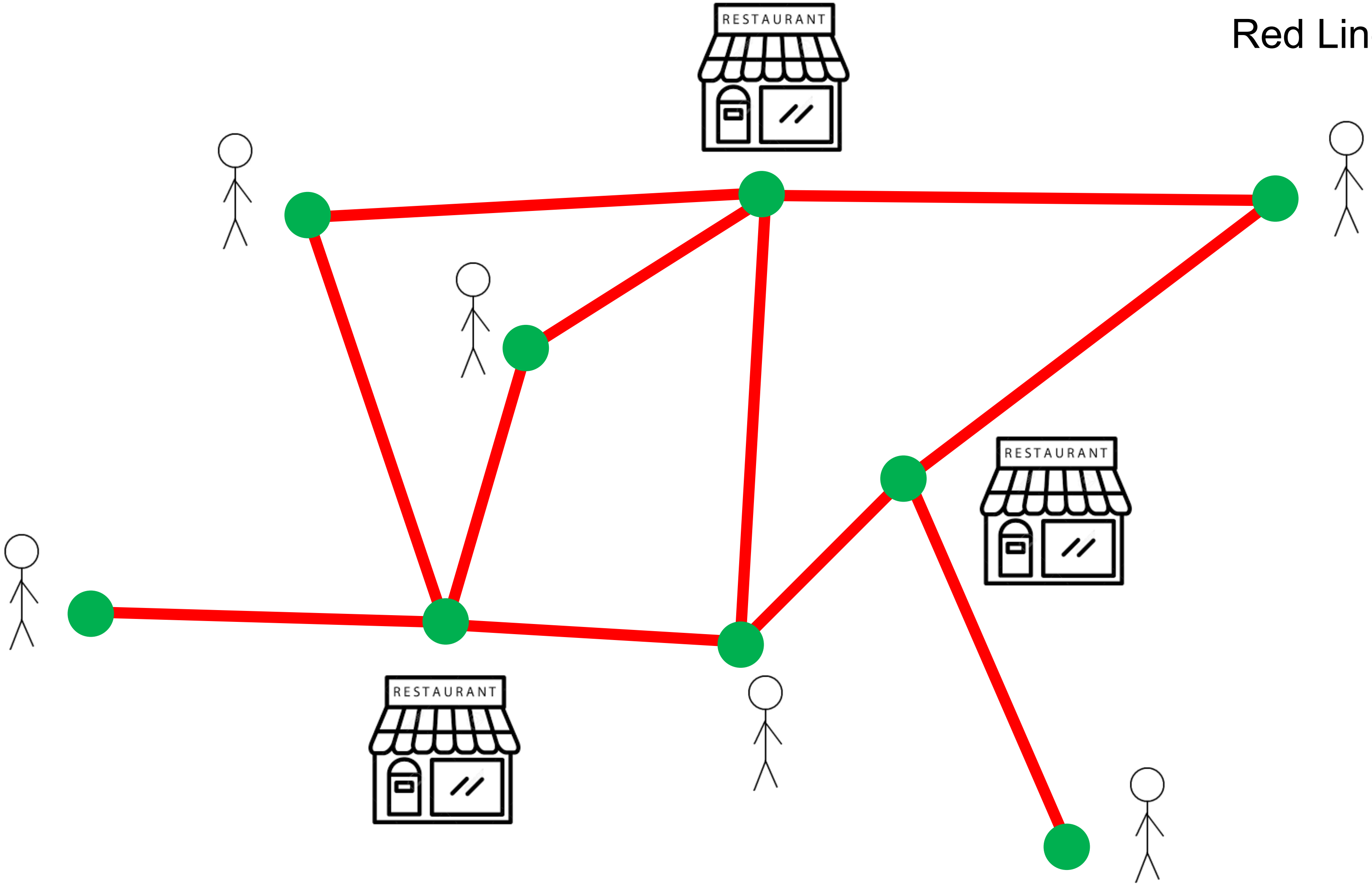


How could we visualize: Restaurants and Potential Customers?

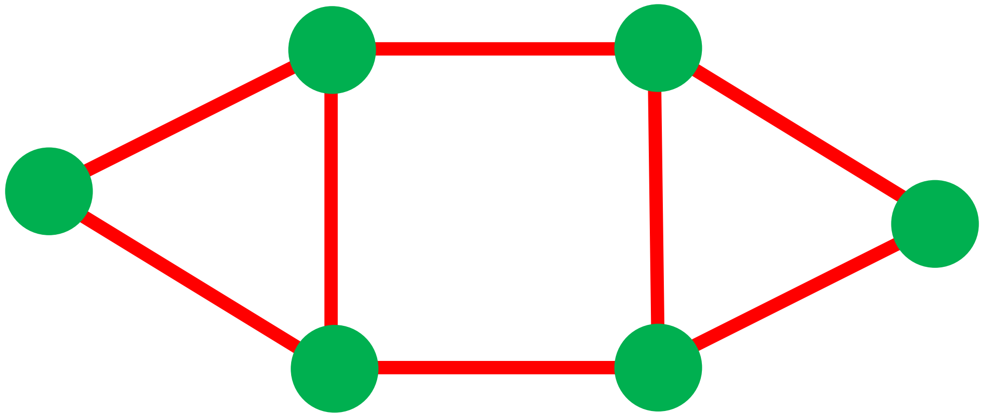


How could we visualize: Restaurants and Potential Customers?

Red Lines = "Would dine at"

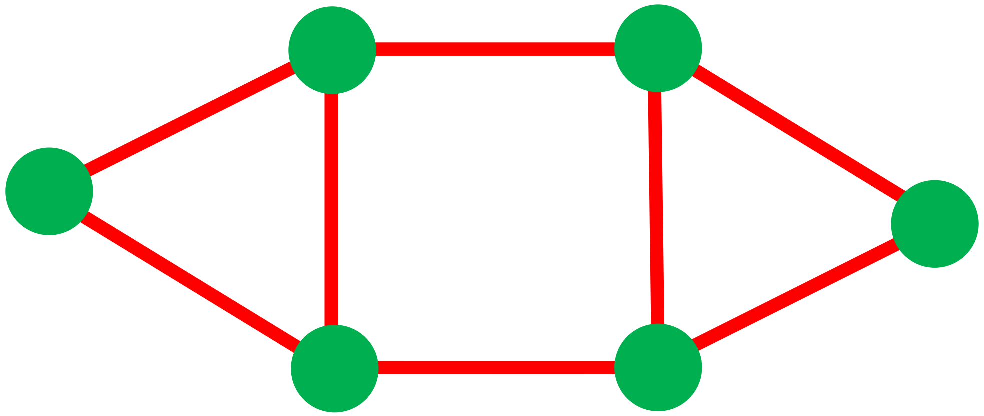


Graphs



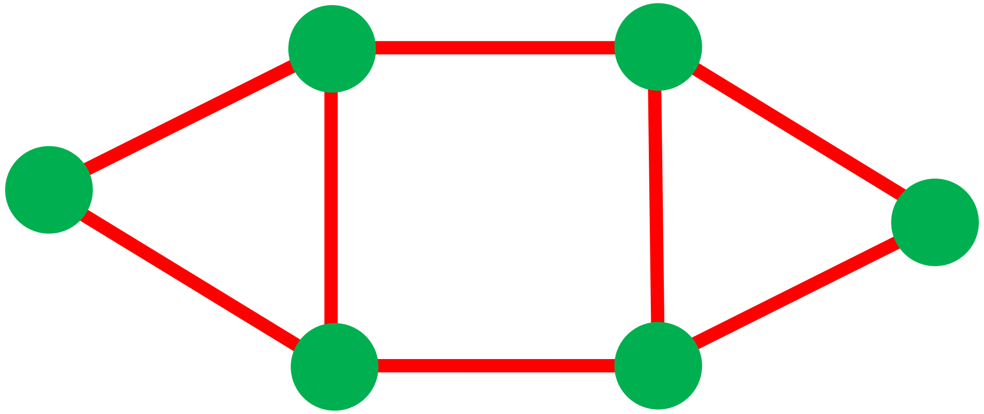
Vertices (or Nodes)

Graphs



Vertices (or Nodes)
Edges

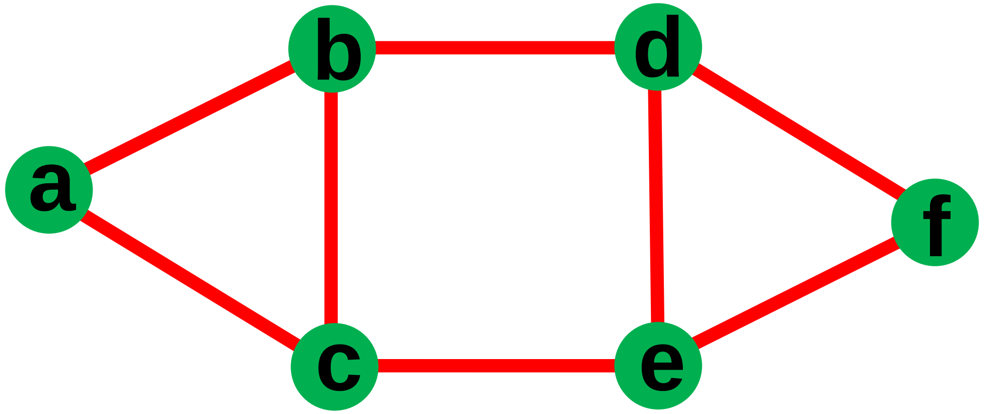
Graphs



Vertices (or Nodes)
Edges

$$\left.\begin{array}{l}\text{Vertices (or Nodes)}\\ \text{Edges}\end{array}\right\} \boldsymbol{G = (V, E)}$$

Graphs

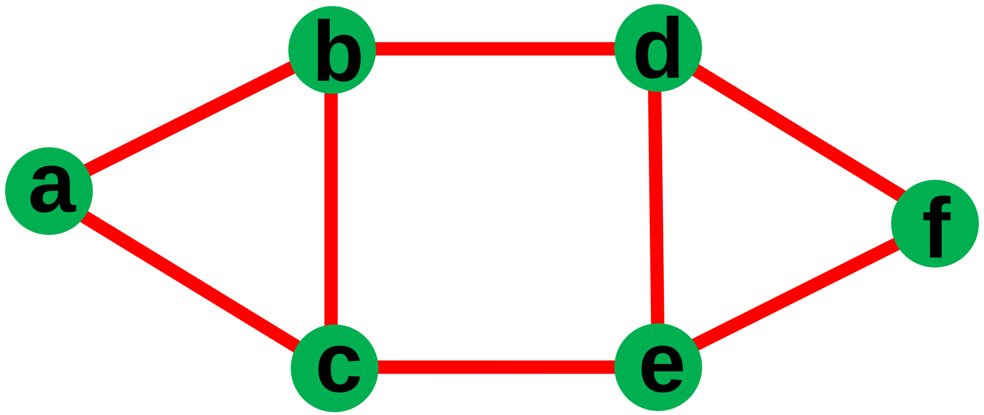


Vertices (or Nodes)
Edges

$$G = (V, E)$$

V = {a, b, c, d, e, f}
E = {(a,b), (a,c), (b,c), (b,d), (c,e), (d,e), (d,f), (e,f)}

Graphs



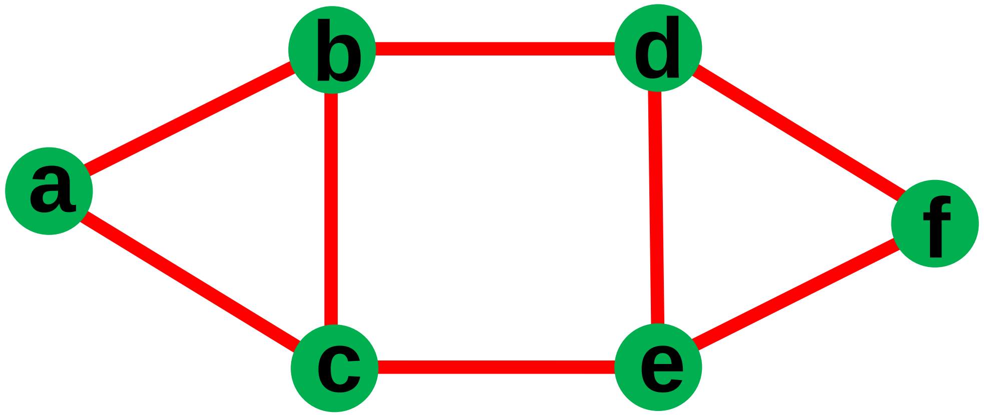
Vertices (or Nodes)
Edges

$\Big\}\ \boldsymbol{G = (V, E)}$

$\boldsymbol{G_1 = G_2}$
If and only if $V_1 = V_2$ and $E_1 = E_2$.

$\boldsymbol{V}$ = {a, b, c, d, e, f}
$\boldsymbol{E}$ = {(a,b), (a,c), (b,c), (b,d), (c,e), (d,e), (d,f), (e,f)}

Graphs

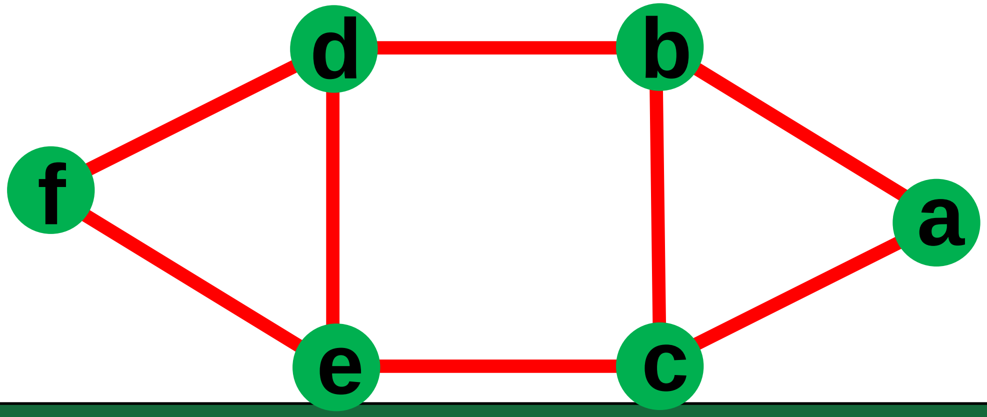


Vertices (or Nodes)
Edges

$\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$

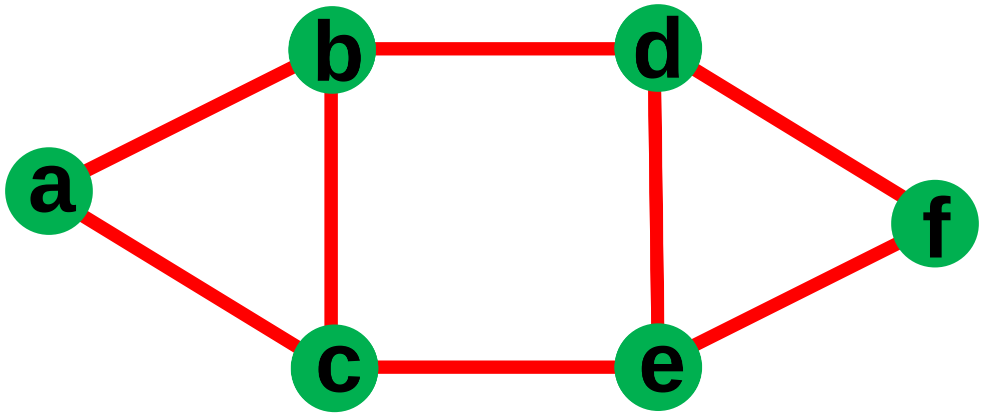
$\boldsymbol{G_1} = \boldsymbol{G_2}$
If and only if $V_1 = V_2$ and $E_1 = E_2$.

$\boldsymbol{V}$ = {a, b, c, d, e, f}
$\boldsymbol{E}$ = {(a,b), (a,c), (b,c), (b,d), (c,e), (d,e), (d,f), (e,f)}



Same graph? ✓

Graphs



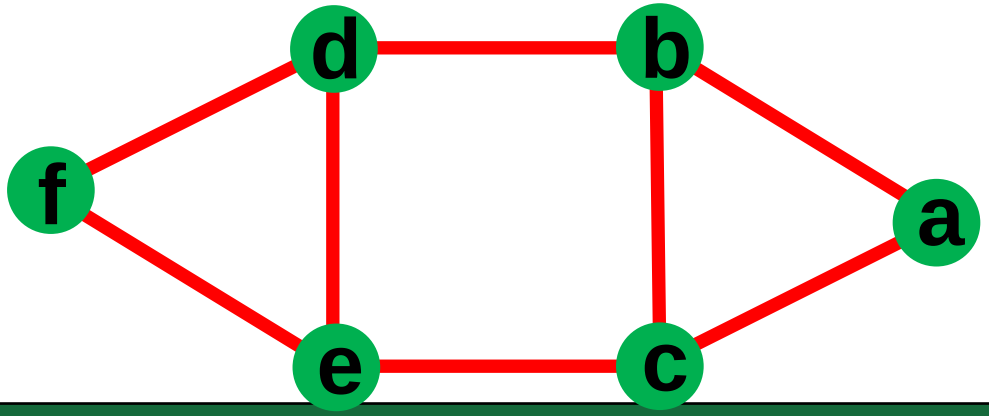
Vertices (or Nodes)
Edges

$$\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$$

$$\boldsymbol{G_1} = \boldsymbol{G_2}$$

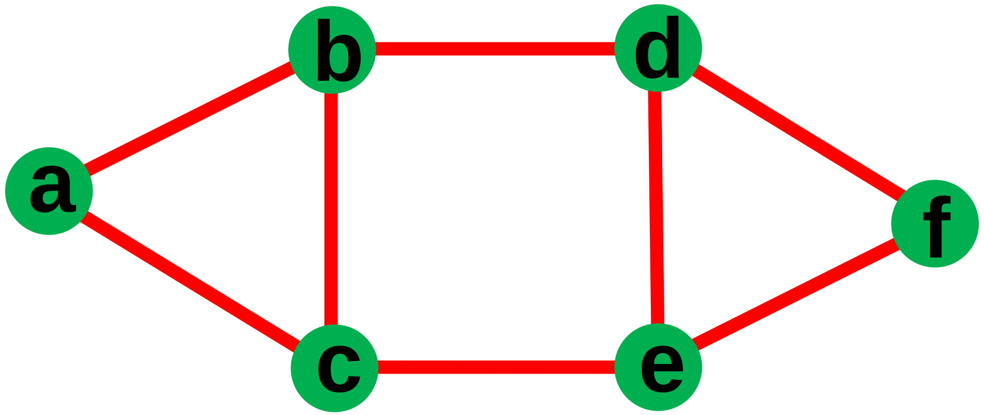
If and only if $V_1 = V_2$ and $E_1 = E_2$.

$\boldsymbol{V}$ = {a, b, c, d, e, f}
$\boldsymbol{E}$ = {(a,b), (a,c), (b,c), (b,d), (c,e), (d,e), (d,f), (e,f)}



Same graph? ✓

Graphs



Vertices (or Nodes)
Edges

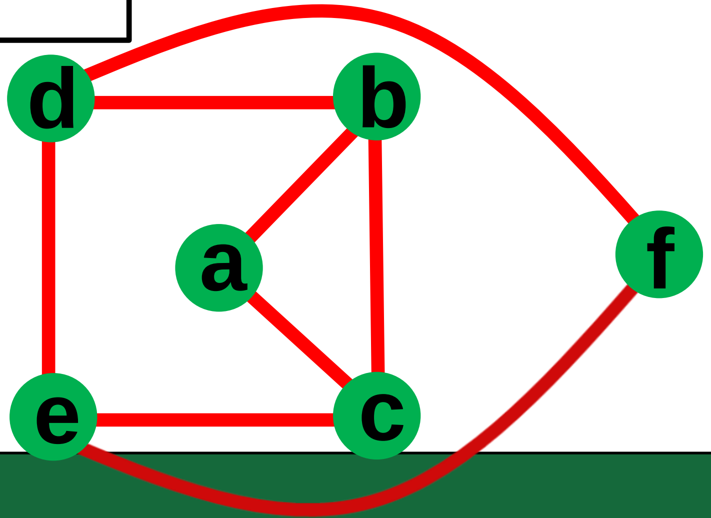
$G = (V, E)$

V = {a, b, c, d, e, f}

E = {(a,b), (a,c), (b,c), (b,d), (c,e), (d,e), (d,f), (e,f)}

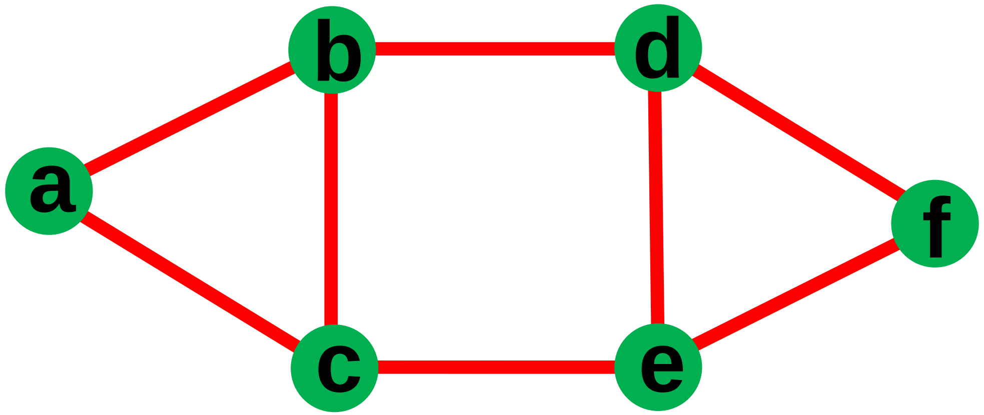
$G_1 = G_2$

If and only if $V_1 = V_2$ and $E_1 = E_2$.



Same graph?

Graphs



Vertices (or Nodes)
Edges

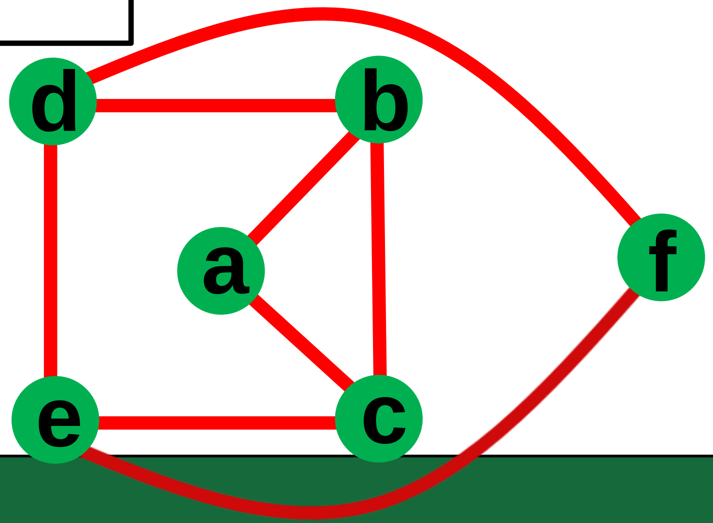
$$\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$$

$$\boldsymbol{G_1} = \boldsymbol{G_2}$$

If and only if $V_1 = V_2$ and $E_1 = E_2$.

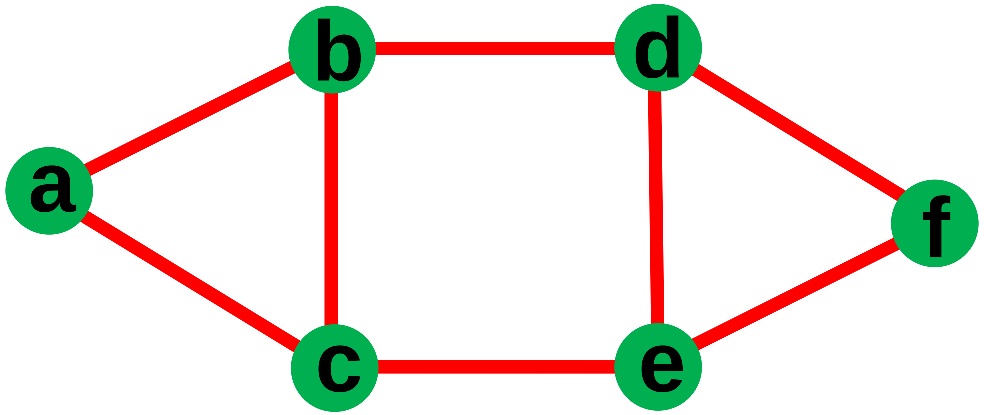
$\boldsymbol{V}$ = {a, b, c, d, e, f}

$\boldsymbol{E}$ = {(a,b), (a,c), (b,c), (b,d), (c,e), (d,e), (d,f), (e,f)}



Same graph? ✓

Graphs



Vertices (or Nodes)
Edges

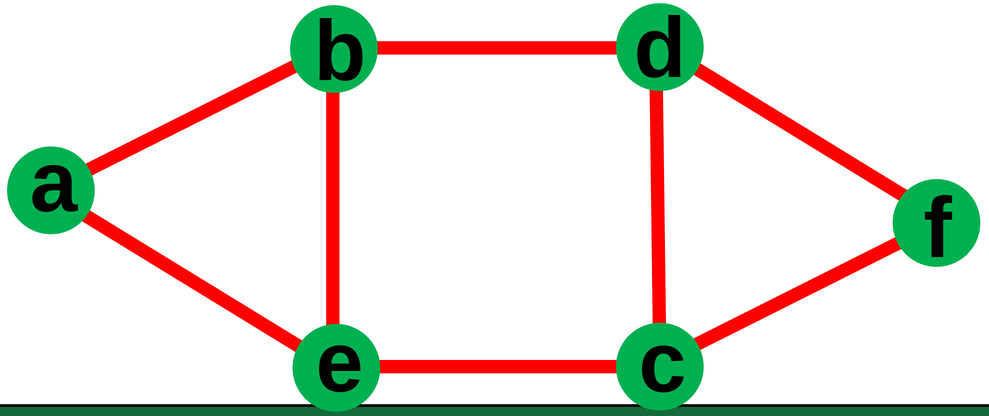
$G = (V, E)$

$$G_1 = G_2$$

If and only if $V_1 = V_2$ and $E_1 = E_2$.

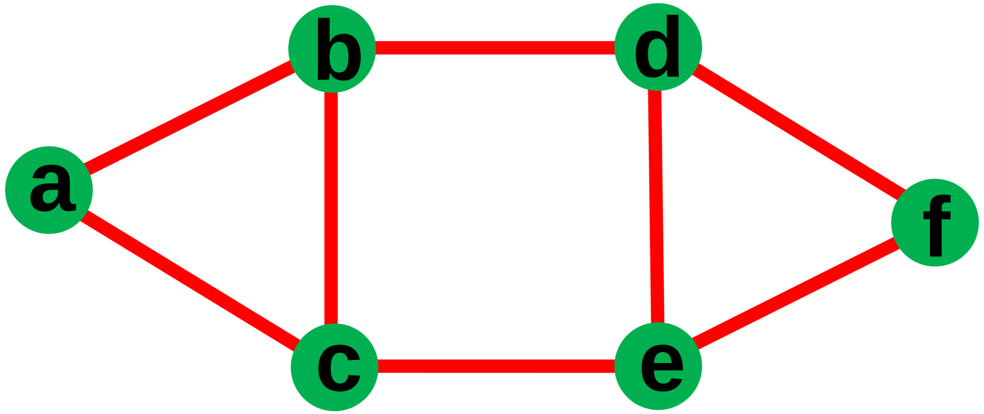
V = {a, b, c, d, e, f}

E = {(a,b), (a,c), (b,c), (b,d), (c,e), (d,e), (d,f), (e,f)}



Same graph?

Graphs

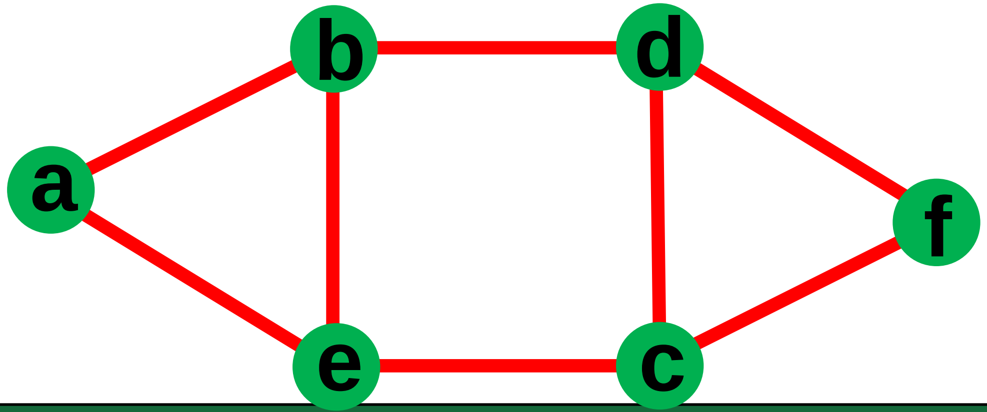


Vertices (or Nodes)
Edges

$\Big\} \; \boldsymbol{G = (V, E)}$

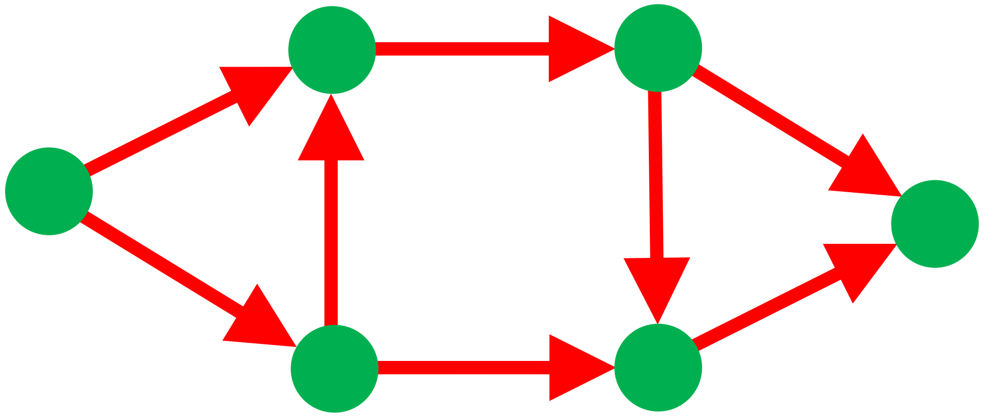
$\boldsymbol{G_1 = G_2}$
If and only if $V_1 = V_2$ and $E_1 = E_2$.

$\boldsymbol{V}$ = {a, b, c, d, e, f}
$\boldsymbol{E}$ = {(a,b), (a,c), (b,c), (b,d), (c,e), (d,e), (d,f), (e,f)}



Same graph? ✗

Graphs

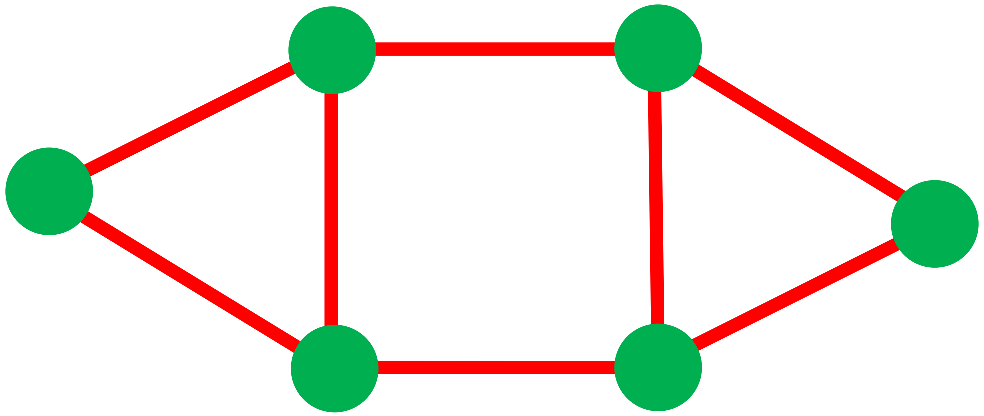


Vertices (or Nodes)
Edges

$G = (V, E)$

- Edges can be directed…

Graphs

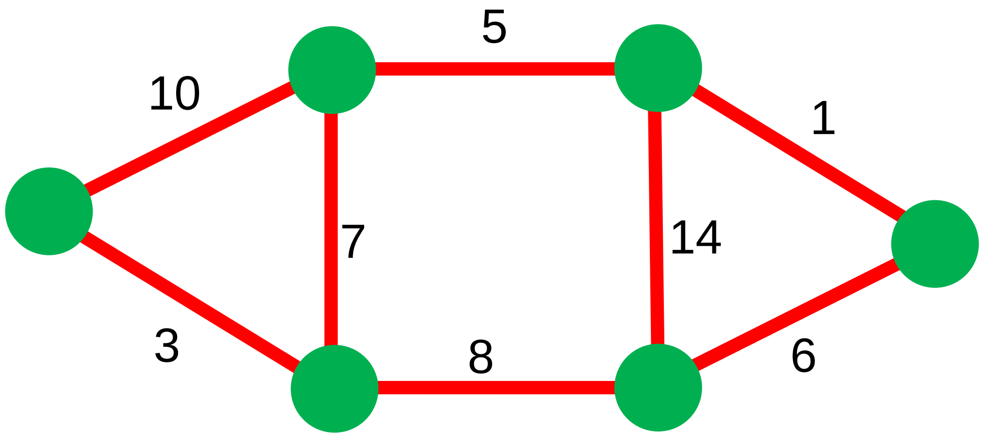


Vertices (or Nodes)
Edges

$$\left.\begin{array}{l}\text{Vertices (or Nodes)}\\ \text{Edges}\end{array}\right\} \boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$$

- Edges can be directed or <u>undirected</u>.

Graphs

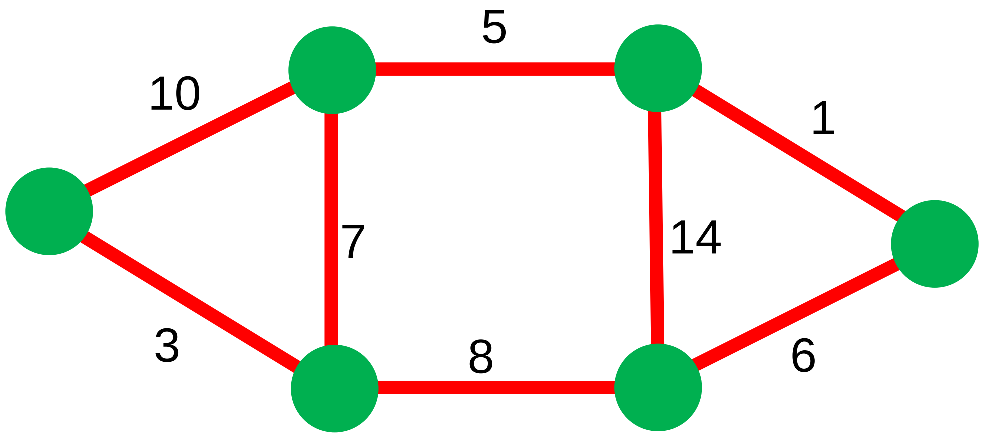


Vertices (or Nodes)
Edges

$$G = (V, E)$$

- Edges can be directed or undirected.
- Edges can have **weights**

Graphs

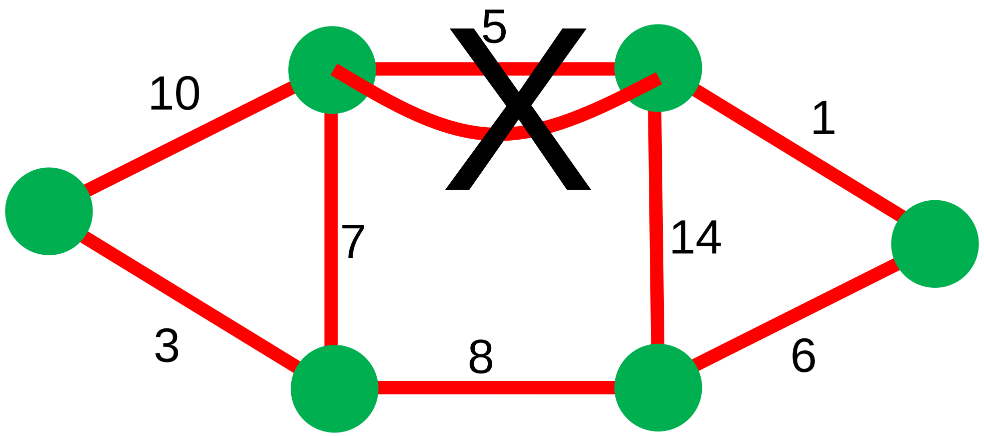


Vertices (or Nodes)
Edges

$$\left.\begin{array}{l}\text{Vertices (or Nodes)}\\ \text{Edges}\end{array}\right\} G = (V, E)$$

- Edges can be directed or undirected.
- Edges can have **weights**
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.

Graphs

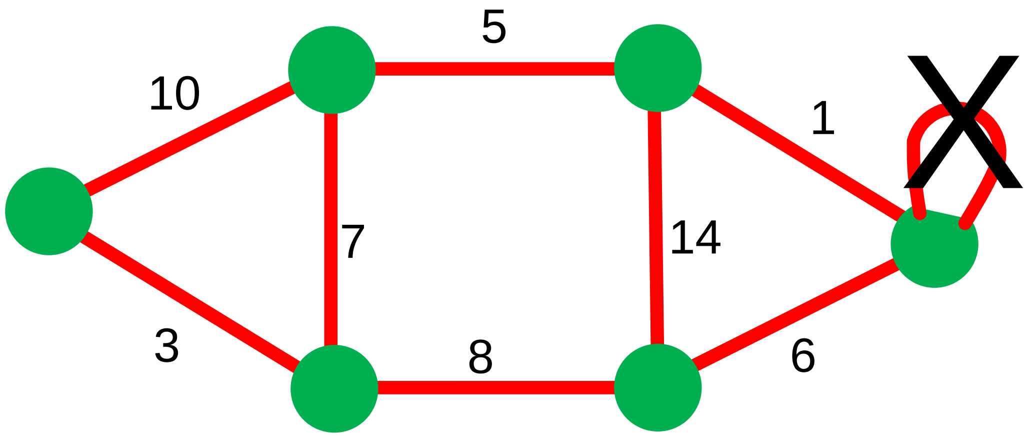


Vertices (or Nodes)
Edges

$\left.\right\} \boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$

- Edges can be directed or <u>undirected</u>.
- Edges can have **weights**
- <u>Simple graph</u> = At most one edge between pair of vertices and no edges that start and end at same vertex.

Graphs

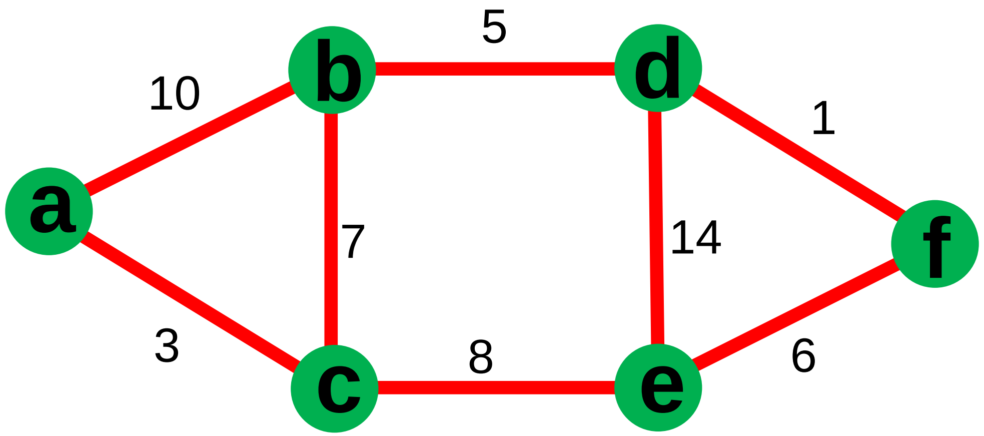


Vertices (or Nodes)
Edges

$$\left.\right\} G = (V, E)$$

- Edges can be directed or undirected.
- Edges can have **weights**
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.

Graphs

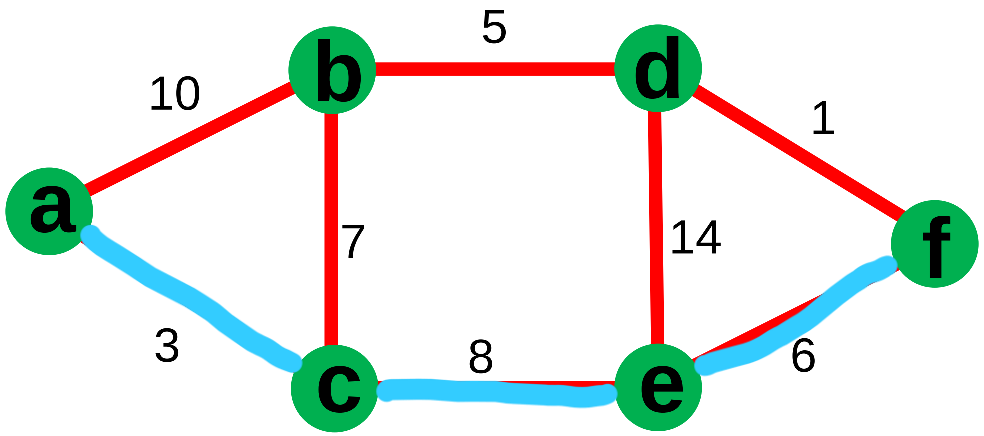


Vertices (or Nodes)
Edges

$G = (V, E)$

- Edges can be directed or undirected.
- Edges can have **weights**
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.

Graphs



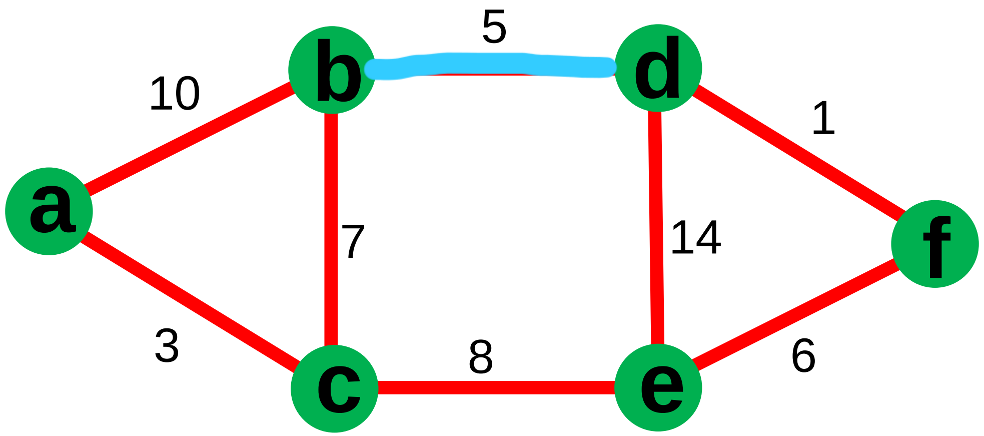
Vertices (or Nodes)
Edges

$$G = (V, E)$$

- Edges can be directed or undirected.
- Edges can have **weights**
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.

a,c,e,f ✓ "cost" of path = 17

Graphs



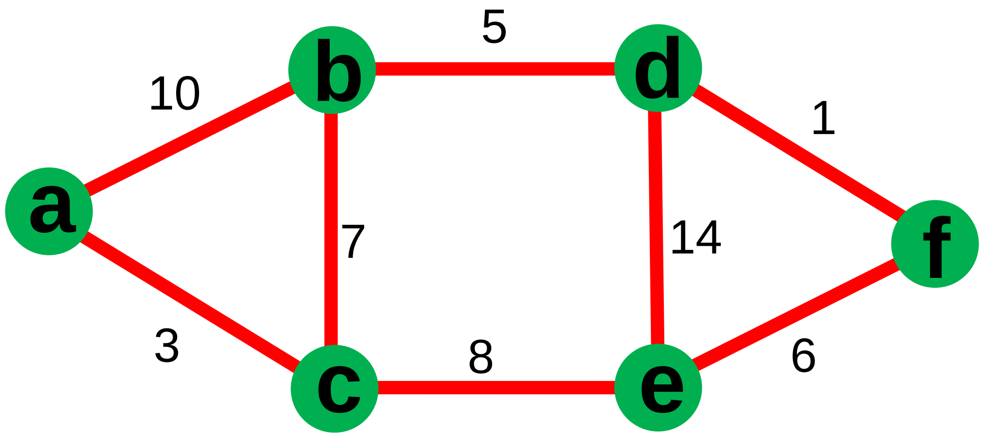
Vertices (or Nodes)
Edges

$$G = (V, E)$$

- Edges can be directed or undirected.
- Edges can have **weights**
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.

b,d ✓

Graphs



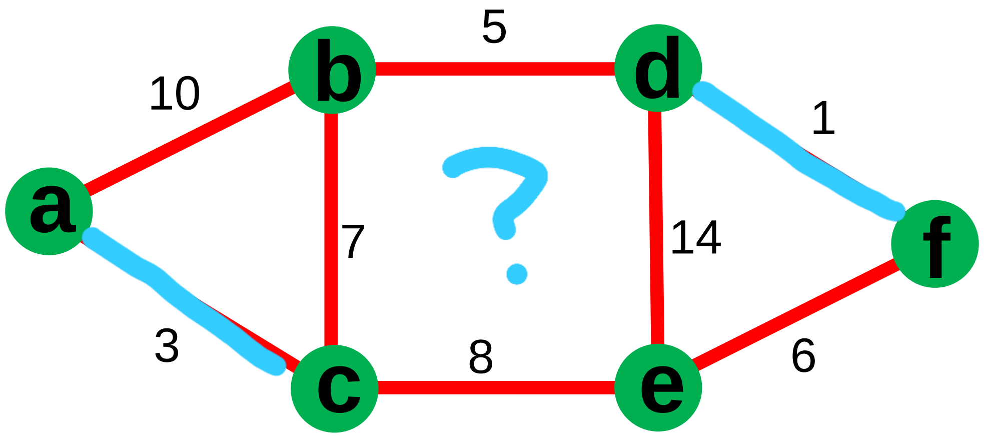
Vertices (or Nodes)
Edges

$$\Big\} \; G = (V, E)$$

- Edges can be directed or undirected.
- Edges can have **weights**
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.

a,c,d,f

Graphs

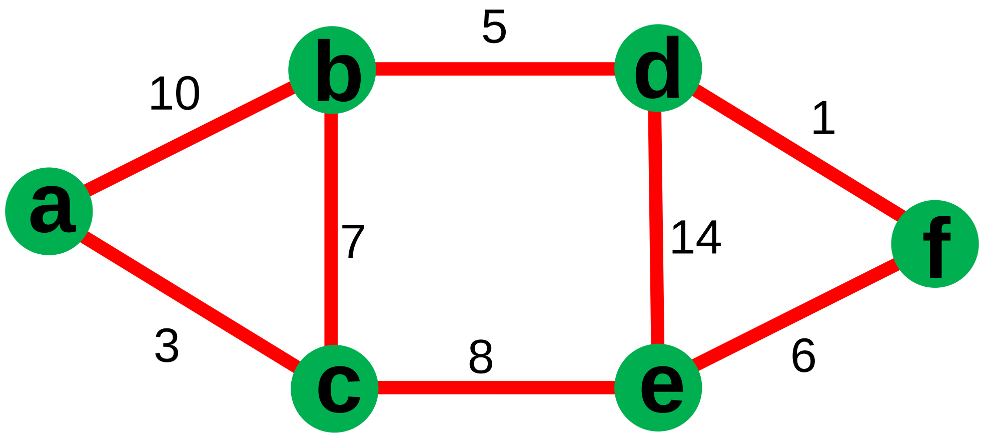


$G = (V, E)$

- Edges can be directed or undirected.
- Edges can have **weights**
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.

a,c,d,f ✗

Graphs



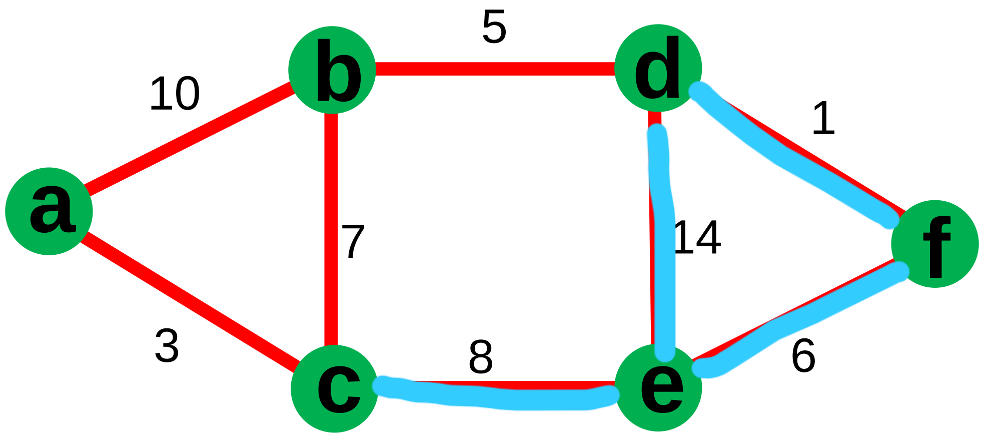
Vertices (or Nodes)
Edges

$$G = (V, E)$$

- Edges can be directed or undirected.
- Edges can have **weights**
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.

c,e,d,f,e

Graphs



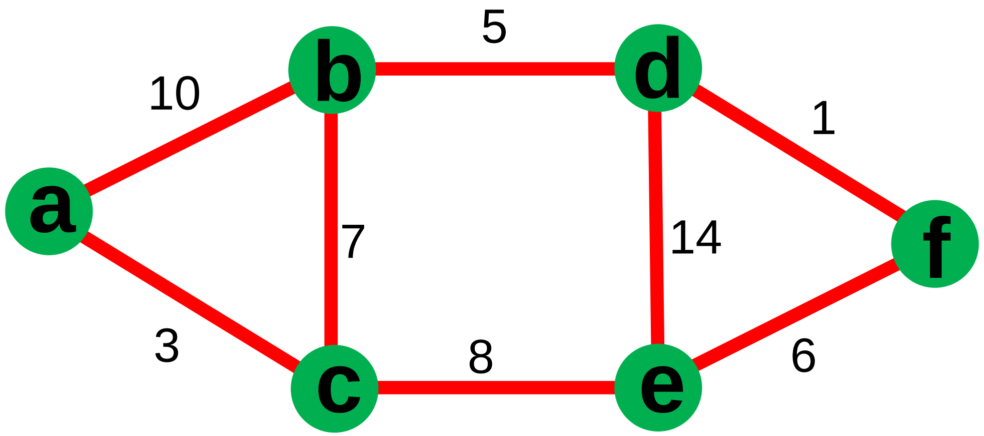
Vertices (or Nodes)
Edges

$$G = (V, E)$$

- Edges can be directed or undirected.
- Edges can have **weights**
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.

c,e,d,f,e ✗

Graphs

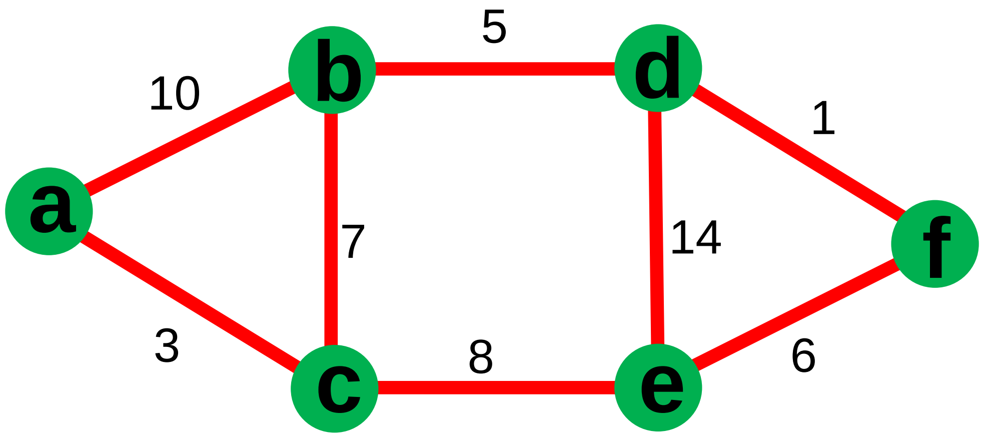


Vertices (or Nodes)
Edges

$$G = (V, E)$$

- Edges can be directed or undirected.
- Edges can have **weights**
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.
- Cycle = Sequence of vertices connected by edge with loop(s).

Graphs

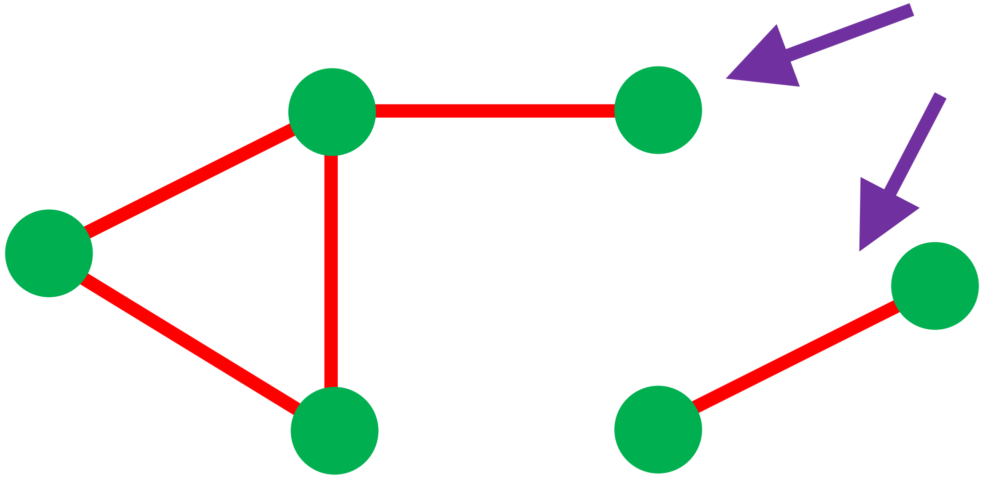


Vertices (or Nodes)
Edges

$$G = (V, E)$$

- Edges can be directed or undirected.
- Edges can have **weights**
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.
- Cycle = Sequence of vertices connected by edge with loop(s).
- Connected Graph = Graph that has a path between every vertex pair.

Graphs



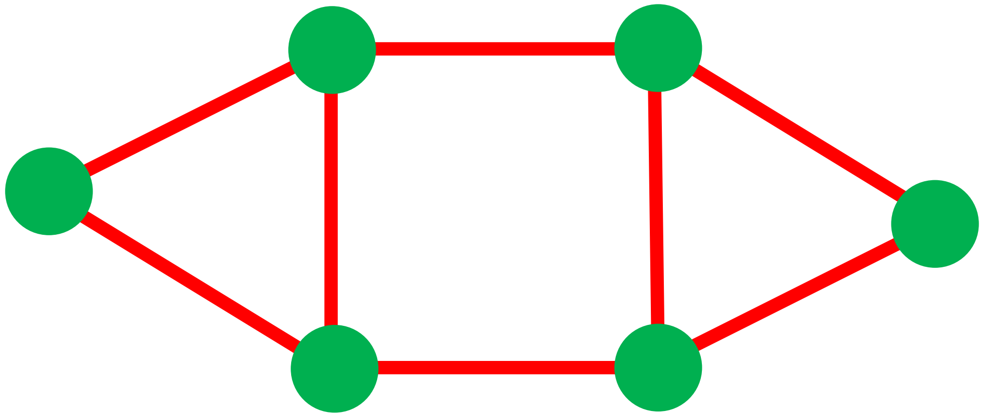
Two Connected Components

Vertices (or Nodes)
Edges

$$\left.\begin{array}{l}\text{Vertices (or Nodes)}\\ \text{Edges}\end{array}\right\} G = (V, E)$$

- Edges can be directed or undirected.
- Edges can have **weights**
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.
- Cycle = Sequence of vertices connected by edge with loop(s).
- Connected Graph = Graph that has a path between every vertex pair.

Graphs

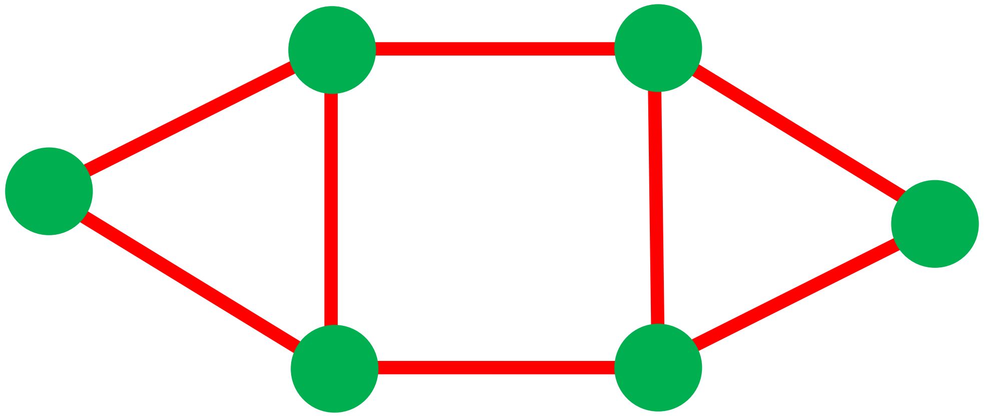


Vertices (or Nodes)
Edges

$$G = (V, E)$$

- Edges can be directed or undirected.
- Edges can have **weights**
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.
- Cycle = Sequence of vertices connected by edge with loop(s).
- Connected Graph = Graph that has a path between every vertex pair.
- Degree of a vertex = $\deg(v)$ = # of edges touching it (undirected).

Graphs



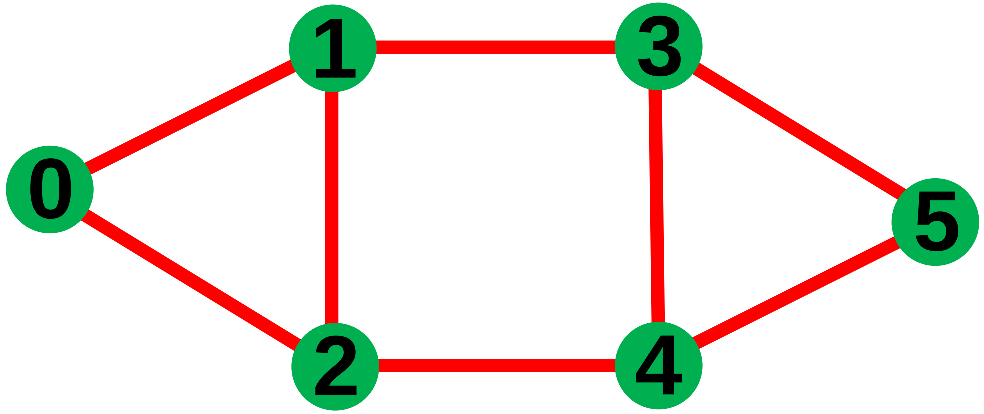
Vertices (or Nodes)
Edges

$$G = (V, E)$$

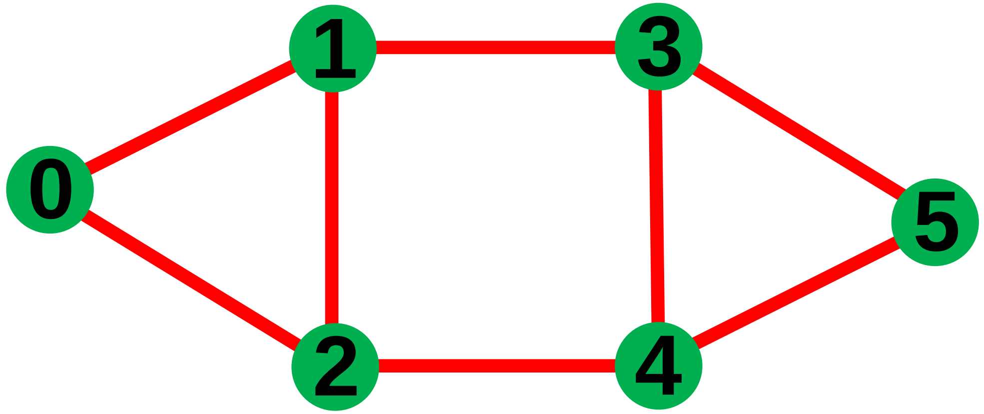
What are some operations we may want to perform on a graph?

- Add vertices/edges.
- Find path between vertex pair.
- Is graph connected?
- Find degree of vertex.
- Is the graph simple?
- Get number of vertices/edges.
- Get neighbors of vertex.
- Is there a cycle?
- Find max degree of graph.

How can we represent a graph in a computer?



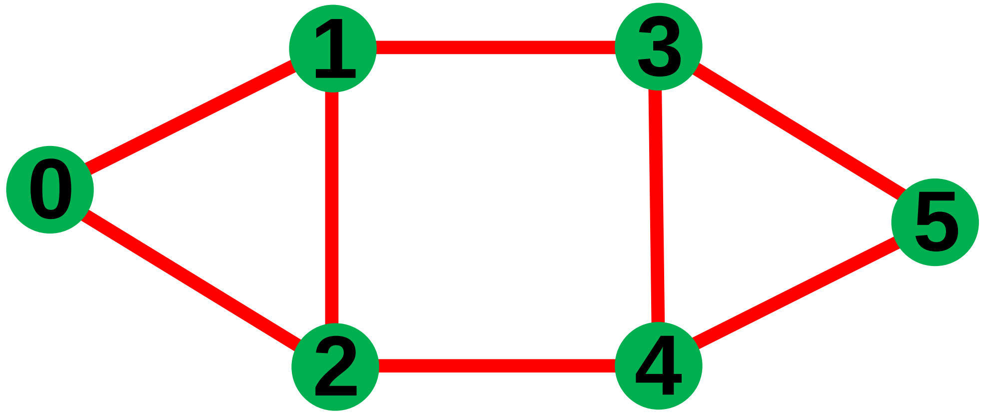
How can we represent a graph in a computer?



1. Adjacency List

0	→	{1,2}
1	→	{0,2,3}
2	→	{0,1,4}
3	→	{1,4,5}
4	→	{2,3,5}
5	→	{3,4}

How can we represent
a graph in a computer?



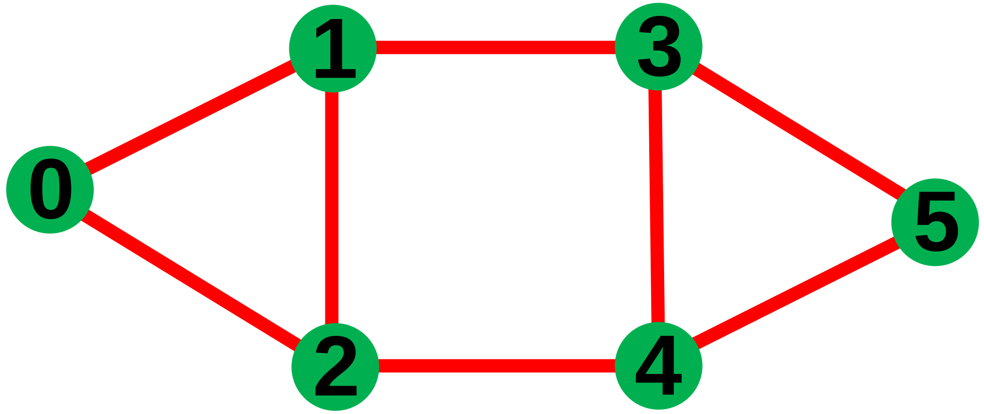
1. Adjacency List

0	→	{1,2}
1	→	{0,2,3}
2	→	{0,1,4}
3	→	{1,4,5}
4	→	{2,3,5}
5	→	{3,4}

2. Adjacency Matrix

	0	1	2	3	4	5
0	F	T	T	F	F	F
1	T	F	T	T	F	F
2	T	T	F	F	T	F
3	F	T	F	F	T	T
4	F	F	T	T	F	T
5	F	F	F	T	T	F

How can we represent
a graph in a computer?



1. Adjacency List

0	→	{1,2}
1	→	{0,2,3}
2	→	{0,1,4}
3	→	{1,4,5}
4	→	{2,3,5}
5	→	{3,4}

2. Adjacency Matrix

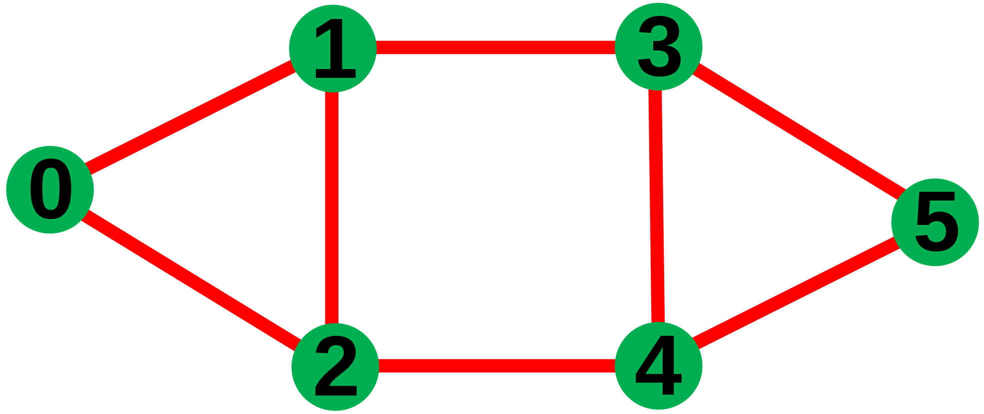
	0	1	2	3	4	5
0	F	T	T	F	F	F
1	T	F	T	T	F	F
2	T	T	F	F	T	F
3	F	T	F	F	T	T
4	F	F	T	T	F	T
5	F	F	F	T	T	F

3. Objects

```
public class Node {
    private Set<Node> neighbors;

    …
}
```

How can we represent a graph in a computer?



1. Adjacency List

0	→	{1,2}
1	→	{0,2,3}
2	→	{0,1,4}
3	→	{1,4,5}
4	→	{2,3,5}
5	→	{3,4}

2. Adjacency Matrix

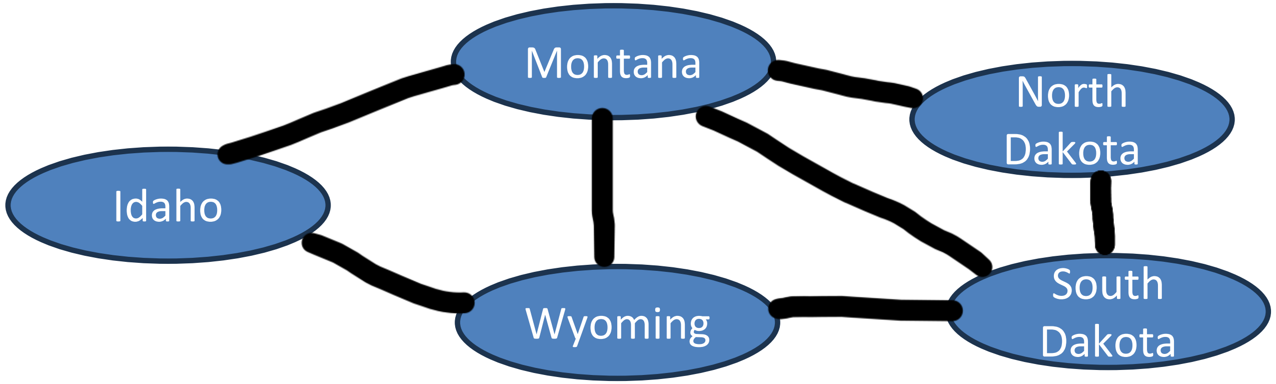
	0	1	2	3	4	5
0	F	T	T	F	F	F
1	T	F	T	T	F	F
2	T	T	F	F	T	F
3	F	T	F	F	T	T
4	F	F	T	T	F	T
5	F	F	F	T	T	F

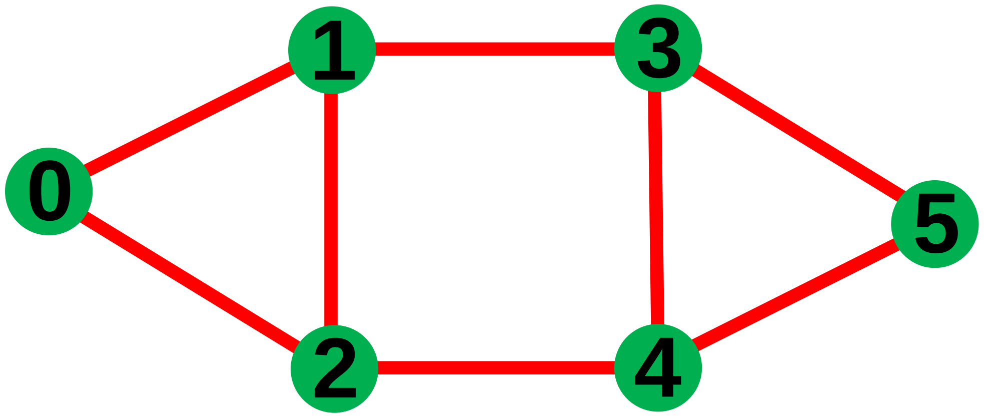
3. Objects

```
public class Node {
   private Set<Node> neighbors;

   …
}
```

1. Adjacency Lists



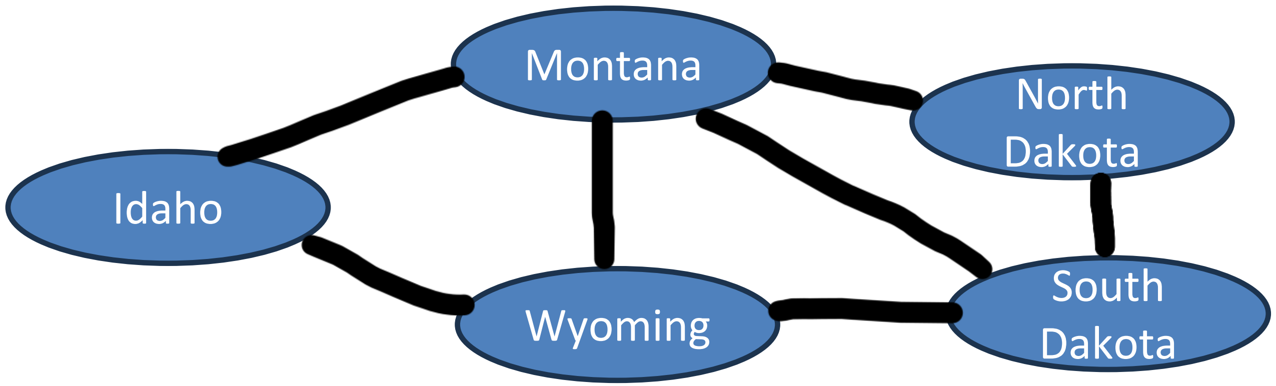


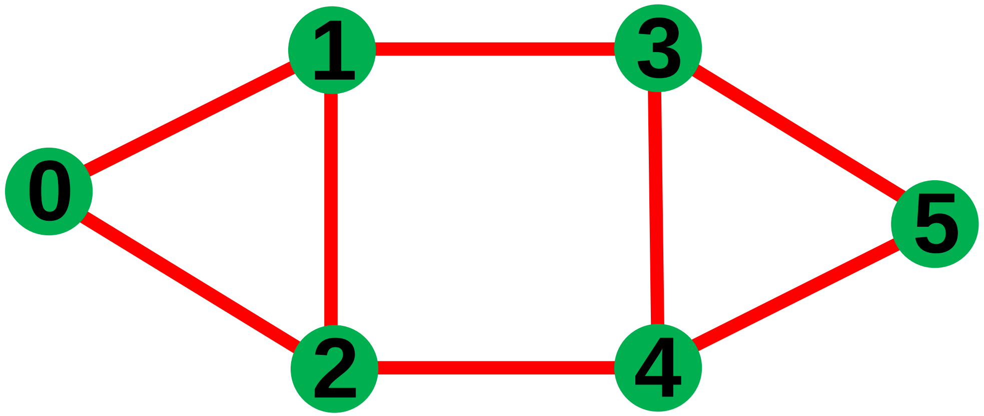
0	→ {1,2}
1	→ {0,2,3}
2	→ {0,1,4}
3	→ {1,4,5}
4	→ {2,3,5}
5	→ {3,4}

Montana: [North Dakota, South Dakota, Wyoming, Idaho]
Idaho: [Montana, Wyoming]
Wyoming: [Idaho, Montana, South Dakota]
North Dakota: [Montana, South Dakota]
South Dakota: [Wyoming, Montana, North Dakota]

?

1. Adjacency Lists



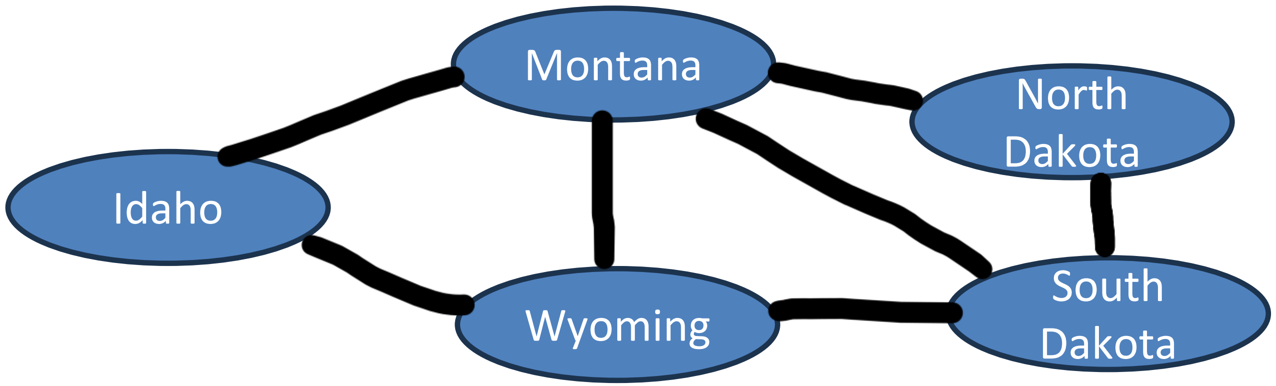


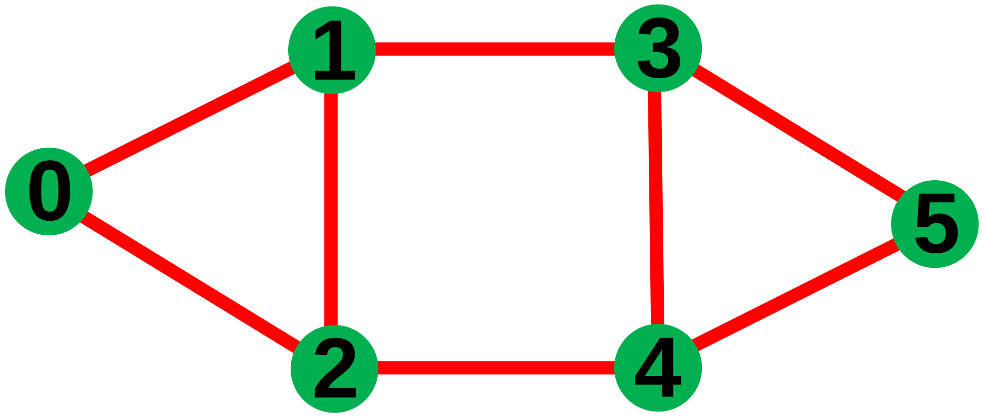
0	→	{1,2}
1	→	{0,2,3}
2	→	{0,1,4}
3	→	{1,4,5}
4	→	{2,3,5}
5	→	{3,4}

Montana: [North Dakota, South Dakota, Wyoming, Idaho]
Idaho: [Montana, Wyoming]
Wyoming: [Idaho, Montana, South Dakota]
North Dakota: [Montana, South Dakota]
South Dakota: [Wyoming, Montana, North Dakota]

HashMap<String, LinkedList<String>>
HashMap<String, LinkedList<Edge>>

1. Adjacency Lists



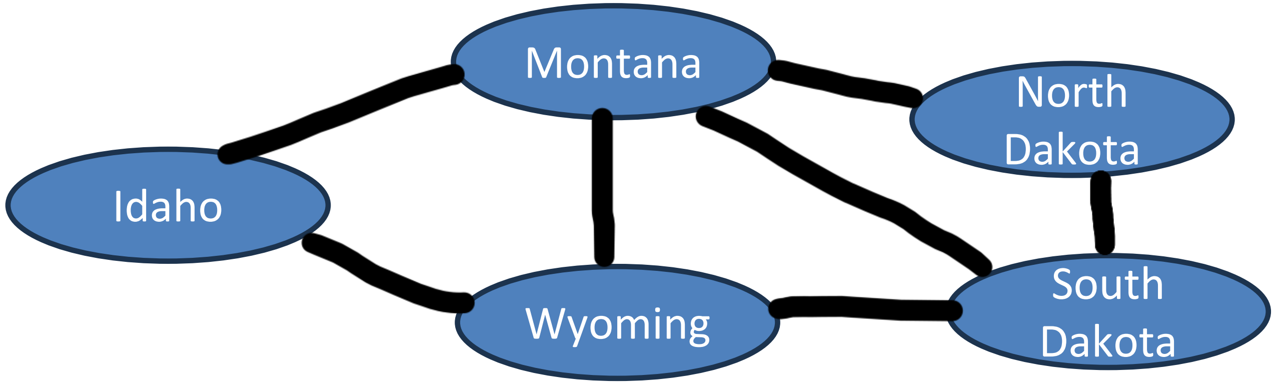


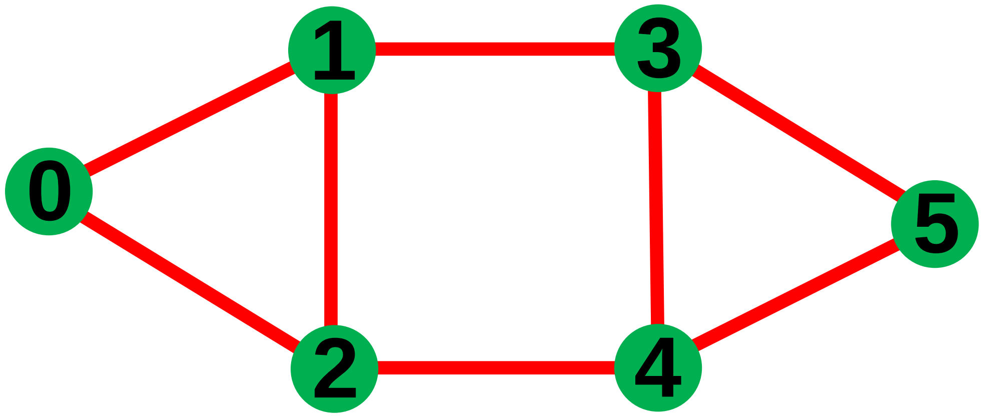
0	→ {1,2} ?
1	→ {0,2,3}
2	→ {0,1,4}
3	→ {1,4,5}
4	→ {2,3,5}
5	→ {3,4}

Montana: [North Dakota, South Dakota, Wyoming, Idaho]
Idaho: [Montana, Wyoming]
Wyoming: [Idaho, Montana, South Dakota]
North Dakota: [Montana, South Dakota]
South Dakota: [Wyoming, Montana, North Dakota]

HashMap<String, LinkedList<String>>
HashMap<String, LinkedList<Edge>>

1. Adjacency Lists





Montana: [North Dakota, South Dakota, Wyoming, Idaho]
Idaho: [Montana, Wyoming]
Wyoming: [Idaho, Montana, South Dakota]
North Dakota: [Montana, South Dakota]
South Dakota: [Wyoming, Montana, North Dakota]

HashMap<String, LinkedList<String>>
HashMap<String, LinkedList<Edge>>

0	→ {1,2}
1	→ {0,2,3}
2	→ {0,1,4}
3	→ {1,4,5}
4	→ {2,3,5}
5	→ {3,4}

Array of Linked Lists