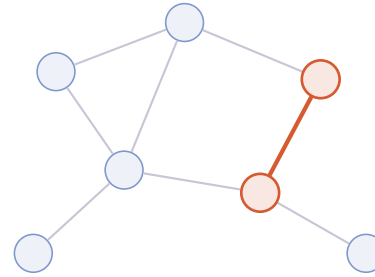
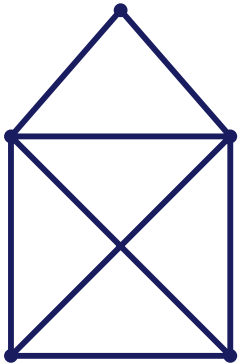
Graphs



What's left when you keep only the connections

Sourav Chakraborty · Graduate TA
CSCI 2270 Fall 2026 · University of Colorado Boulder

While you settle in



Warm-up puzzle

Draw this house in **one stroke**.

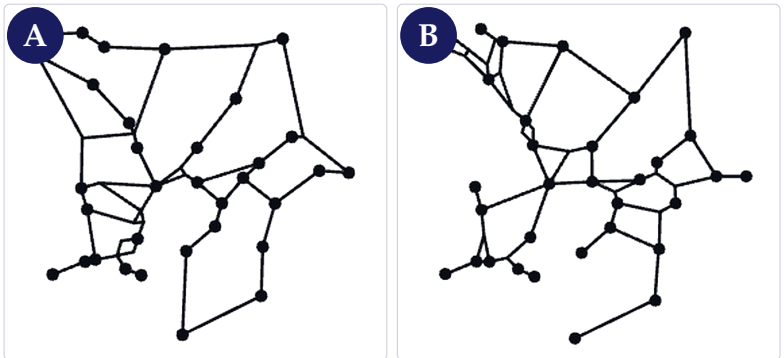
Don't lift your pen.

Don't trace any line twice.

Try a few different starting corners.

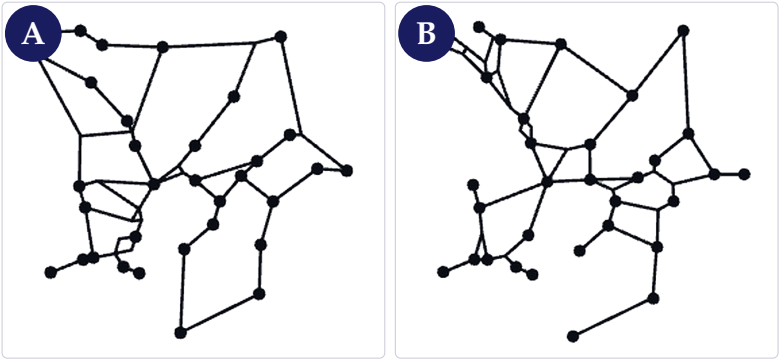
We'll come back to this in about ten minutes, and you'll know why it works.

Two networks



Dots are cities around Tokyo, lines are links. From Tero et al., *Science* 2010.

Two networks

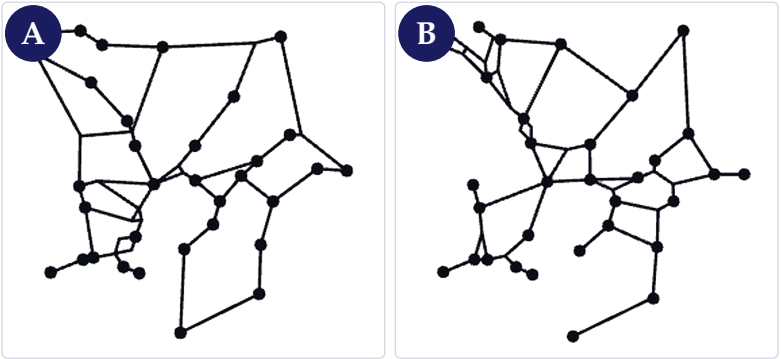


Dots are cities around Tokyo, lines are links. From Tero et al., *Science* 2010.

One was **designed by engineers** over decades.

One was **grown in about a day** by something with no brain.

Two networks



Dots are cities around Tokyo, lines are links. From Tero et al., *Science* 2010.

One was **designed by engineers** over decades.

One was **grown in about a day** by something with no brain.

Which is which?

Hands up: A or B?

Hold that thought.

Today

1 Model

Turn a real problem into a graph: decide what the vertices and the edges are.

2 Speak

Vertex, edge, degree, path, cycle, connected, directed, weighted.

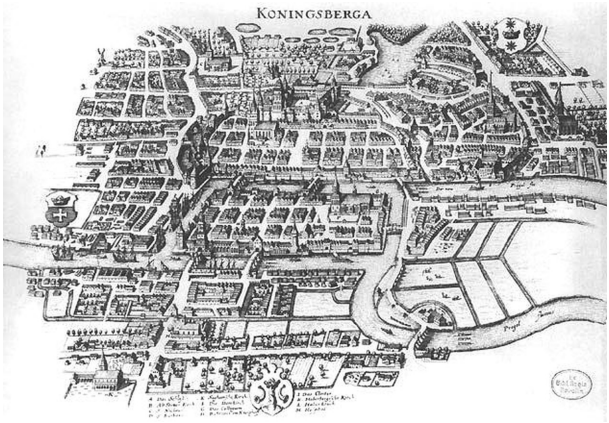
3 Store

Adjacency matrix or adjacency list, and know when to use which.

Route Bridges → Is it a graph? → Vocabulary → Storing graphs → Payoff

? in the corner = a question still open. It goes away when we answer it.

Königsberg, 1736

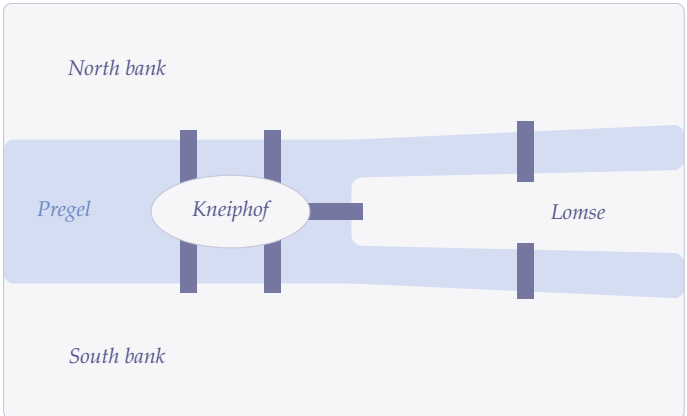


Königsberg in 1652, engraving by Merian. Today it's Kaliningrad, Russia.

A Sunday-walk puzzle

A river city in Prussia, with two islands in the river Pregel.

Königsberg, 1736



The same city, simplified: only land, water and bridges.

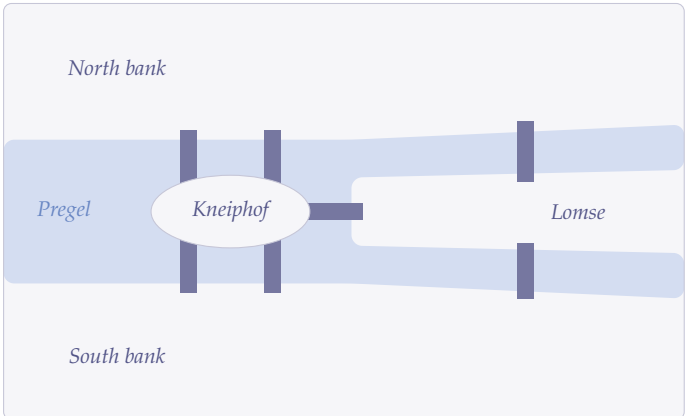
A Sunday-walk puzzle

A river city in Prussia, with two islands in the river Pregel.

It cuts the city into **four pieces of land**, joined by **seven bridges**.

Can you walk through town crossing **every bridge exactly once**?

Königsberg, 1736



The same city, simplified: only land, water and bridges.

A Sunday-walk puzzle

A river city in Prussia, with two islands in the river Pregel.

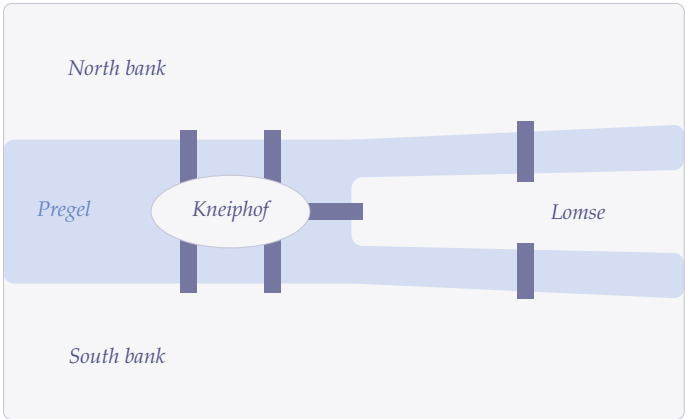
It cuts the city into **four pieces of land**, joined by **seven bridges**.

Can you walk through town crossing **every bridge exactly once**?

Your turn: 60 seconds with a neighbor.

Start anywhere, end anywhere.

Euler's move

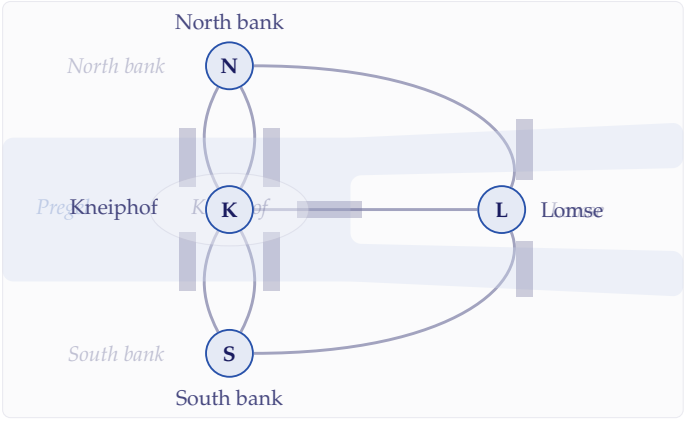


Leonhard Euler, 1736

The shape of the city doesn't matter.



Euler's move



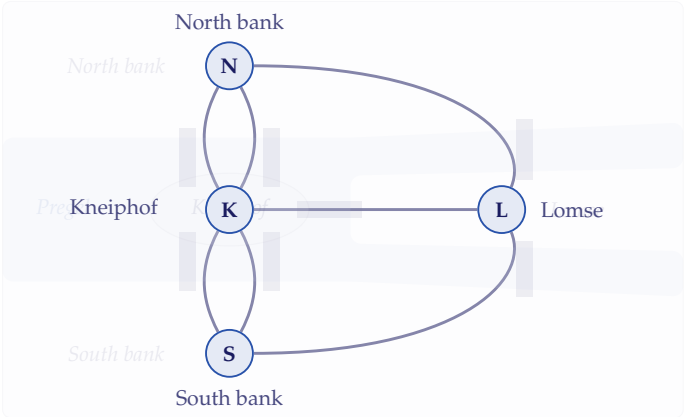
Leonhard Euler, 1736

The shape of the city doesn't matter.



Each piece of land $\rightarrow$ a **point**.

Euler's move



Leonhard Euler, 1736

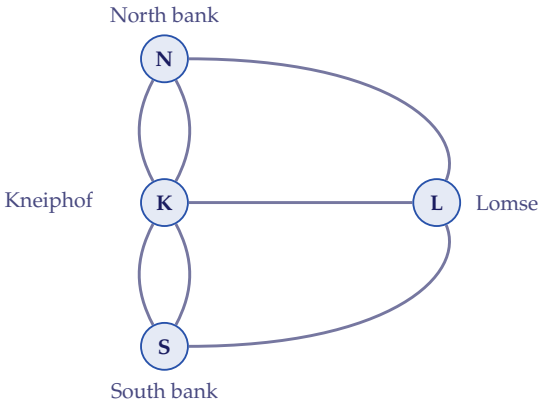


The shape of the city doesn't matter.

Each piece of land $\rightarrow$ a **point**.

Each bridge $\rightarrow$ a **line** between two points.

Euler's move



Leonhard Euler, 1736



The shape of the city doesn't matter.

Each piece of land → a **point**.

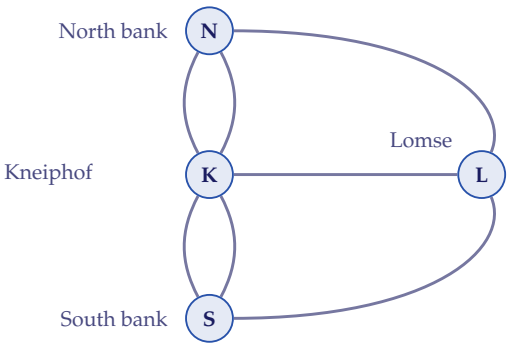
Each bridge → a **line** between two points.

Keep only what's connected to what.

That's a **graph**.

Modeling checklist V = land masses | E = bridges | Directed? | Weighted? | Sparse or dense?

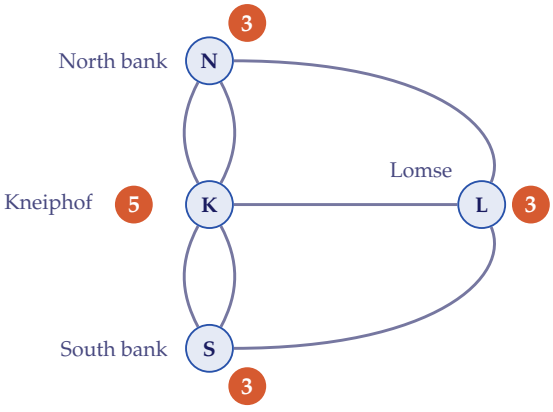
Why nobody could do it



Count the bridges at each piece of land

How many bridges touch N? S? K? L?

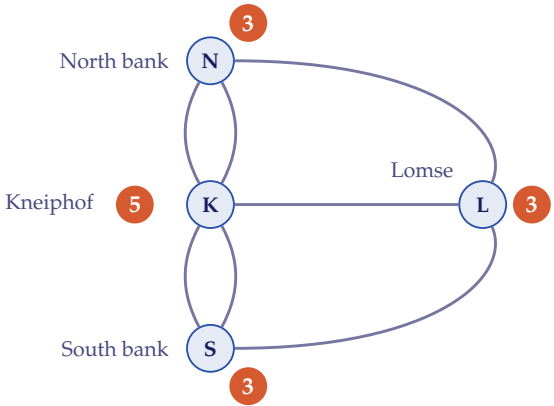
Why nobody could do it



Count the bridges at each piece of land

That count is the vertex's **degree**: K has 5; N, S, L have 3.

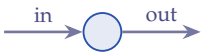
Why nobody could do it



Count the bridges at each piece of land

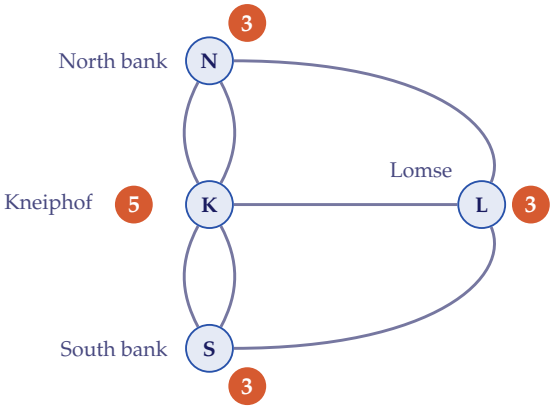
That count is the vertex's **degree**: K has 5; N, S, L have 3.

Walking *through* a piece of land uses two bridges: one in, one out.



So every land you pass through needs an **even** degree. Only the start and the end can be odd.

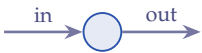
Why nobody could do it



Count the bridges at each piece of land

That count is the vertex's **degree**: K has 5; N, S, L have 3.

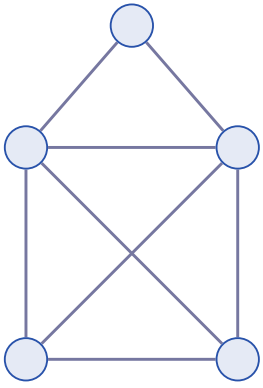
Walking *through* a piece of land uses two bridges: one in, one out.



So every land you pass through needs an **even** degree. Only the start and the end can be odd.

Königsberg has **four** odd vertices. At most two are allowed. **No such walk exists.**

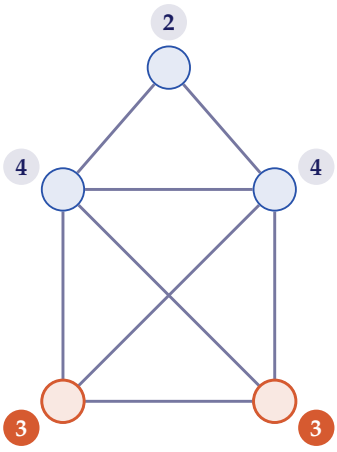
Back to the house



The warm-up, as a graph

5 corners, 8 lines. What are the degrees?

Back to the house

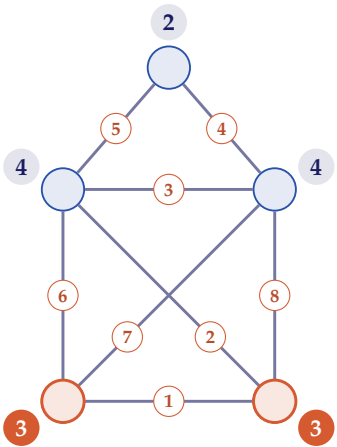


The warm-up, as a graph

5 corners, 8 lines. What are the degrees?

Exactly **two** odd vertices: the bottom corners.
Same rule as Königsberg, opposite verdict.

Back to the house



The warm-up, as a graph

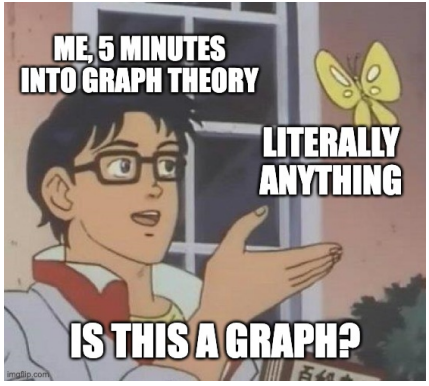
5 corners, 8 lines. What are the degrees?

Exactly **two** odd vertices: the bottom corners.
Same rule as Königsberg, opposite verdict.

So start at one bottom corner and finish at the other. Any other start fails.

German kids know this puzzle: *"Das ist das Haus vom Ni-ko-laus"*, eight syllables for eight strokes.

Is this a graph?



The game for the next seven minutes

I show you something from the real world. You shout out:

What are the **vertices**?

What are the **edges**?

Each new picture adds one more question to the checklist.

Modeling checklist V = ? | E = ? | Directed? | Weighted? | Sparse or dense?

Who is friends with whom?

The world: a friends list

ana	**ana** and **ben** are friends	Friends
ana	**ana** and **cy** are friends	Friends
ben	**ben** and **cy** are friends	Friends
cy	**cy** and **dev** are friends	Friends

model it →

The graph

What are the **vertices**?

What are the **edges**?

Checklist V = ? | E = ? | Directed? | Weighted? | Sparse or dense?

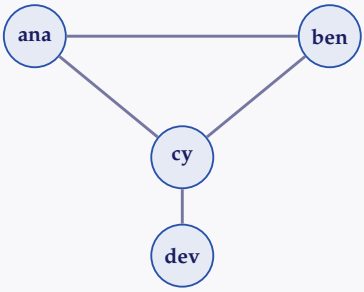
Who is friends with whom?

The world: a friends list

ana	**ana** and **ben** are friends	Friends
ana	**ana** and **cy** are friends	Friends
ben	**ben** and **cy** are friends	Friends
cy	**cy** and **dev** are friends	Friends

model it →

The graph



Checklist V = people | E = friendships | Directed? | Weighted? | Sparse or dense?

Who is friends with whom?

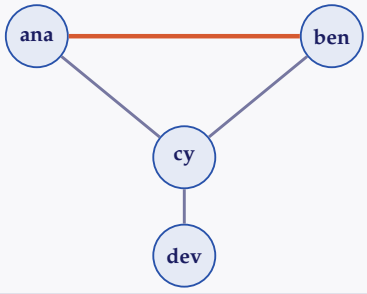
The world: a friends list

ana	**ana** and **ben** are friends	Friends
ana	**ana** and **cy** are friends	Friends
ben	**ben** and **cy** are friends	Friends
cy	**cy** and **dev** are friends	Friends

model it →

The graph

friendship goes both ways: one edge, no arrow



Checklist V = people | E = friendships | Directed? no | Weighted? | Sparse or dense?

Who follows whom?

The world: a follow list

ana	**ana** follows **ben**	Following
ben	**ben** follows **ana**	Following
cy	**cy** follows **ana**	Following
dev	**dev** follows **cy**	Following

model it →

The graph

What are the **vertices**?

What are the **edges**?

Checklist V = ? | E = ? | Directed? | Weighted? | Sparse or dense?

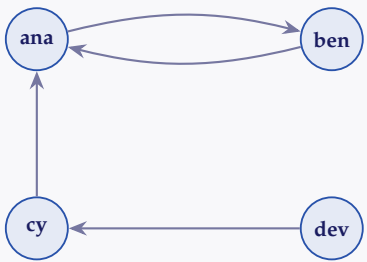
Who follows whom?

The world: a follow list

ana	**ana** follows **ben**	Following
ben	**ben** follows **ana**	Following
cy	**cy** follows **ana**	Following
dev	**dev** follows **cy**	Following

model it →

The graph



Checklist V = people | E = follows | Directed? | Weighted? | Sparse or dense?

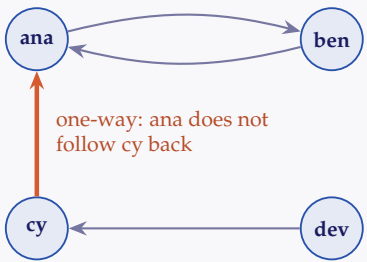
Who follows whom?

The world: a follow list

ana	**ana** follows **ben**	Following
ben	**ben** follows **ana**	Following
cy	**cy** follows **ana**	Following
dev	**dev** follows **cy**	Following

model it →

The graph



Checklist V = people | E = follows | Directed? yes | Weighted? | Sparse or dense?

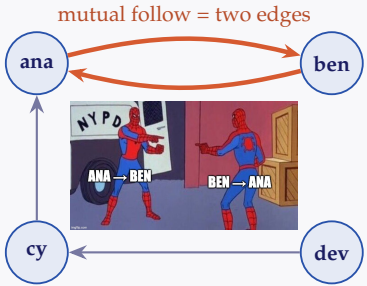
Who follows whom?

The world: a follow list

ana	**ana** follows **ben**	Following
ben	**ben** follows **ana**	Following
cy	**cy** follows **ana**	Following
dev	**dev** follows **cy**	Following

model it →

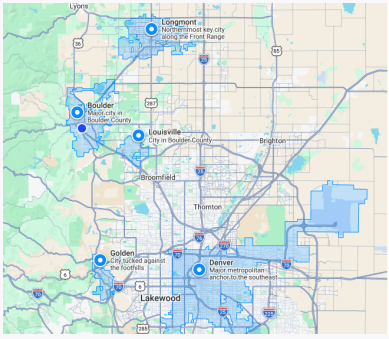
The graph



Checklist V = people | E = follows | Directed? yes | Weighted? | Sparse or dense?

How long to drive there?

The world: a map



model it →

The graph

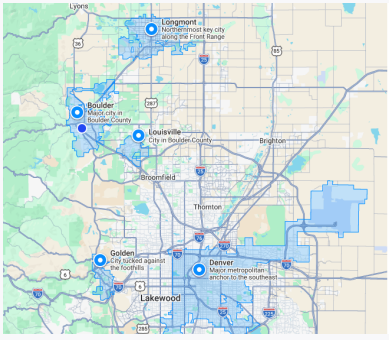
What are the **vertices**?

What are the **edges**?

Checklist V = ? | E = ? | Directed? | Weighted? | Sparse or dense?

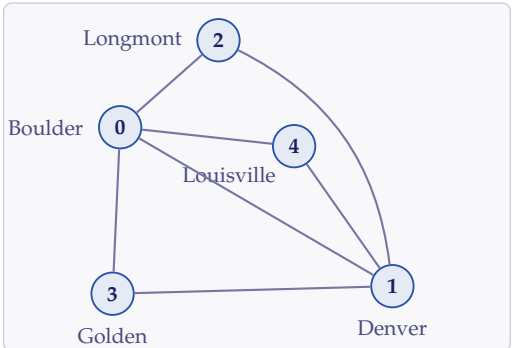
How long to drive there?

The world: a map



model it

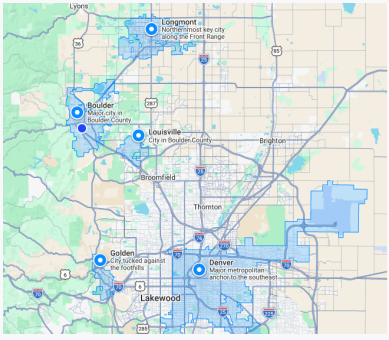
The graph



Checklist V = towns | E = roads | Directed? no | Weighted? | Sparse or dense?

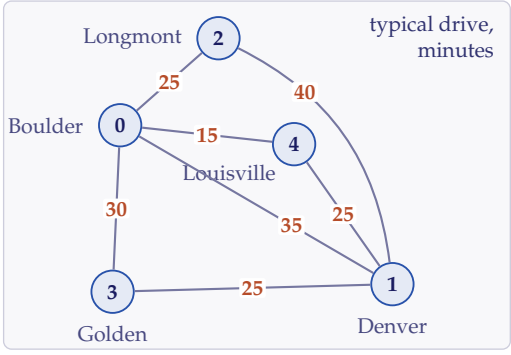
How long to drive there?

The world: a map



model it

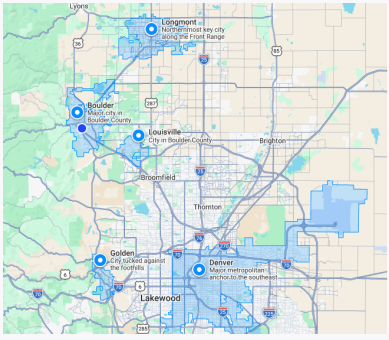
The graph



Checklist V = towns | E = roads | Directed? no | Weighted? minutes | Sparse or dense?

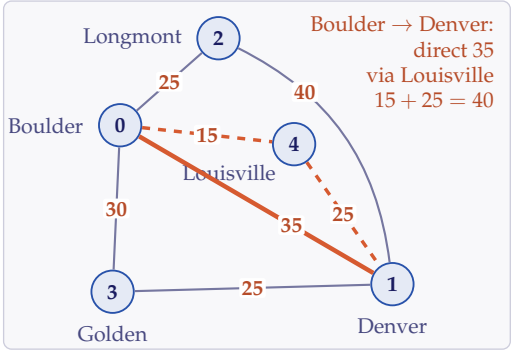
How long to drive there?

The world: a map



model it

The graph



Checklist V = towns | E = roads | Directed? no | Weighted? minutes | Sparse or dense?

What can you take next?

The world: the course catalog

CSCI 2400 Computer Systems
Requires CSCI 2270 or CSCI 2275.

CSCI 3104 Algorithms
Requires (CSCI 2270 or 2275) and (APPM 1360 or MATH 2300) and (CSCI 2824 or …).

CSCI 3753 Operating Systems
Requires (CSCI 2270 or 2275) and (CSCI 2400 or …).

CU Boulder catalog 2026–27, shortened

model it →

The graph

What are the **vertices**?

What are the **edges**?

Checklist V = ? | E = ? | Directed? | Weighted? | Sparse or dense?

What can you take next?

The world: the course catalog

CSCI 2400 Computer Systems
Requires CSCI 2270 or CSCI 2275.

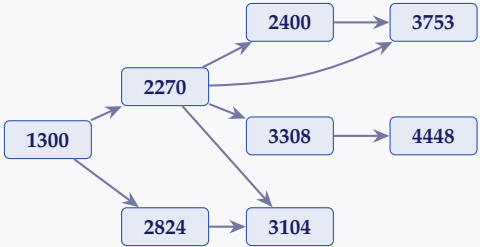
CSCI 3104 Algorithms
Requires (CSCI 2270 or 2275) and (APPM 1360 or MATH 2300) and (CSCI 2824 or …).

CSCI 3753 Operating Systems
Requires (CSCI 2270 or 2275) and (CSCI 2400 or …).

CU Boulder catalog 2026–27, shortened

model it →

The graph



Checklist V = courses | E = "required before" | Directed? yes | Weighted? no | Sparse or dense?

What can you take next?

The world: the course catalog

CSCI 2400 Computer Systems
Requires CSCI 2270 or CSCI 2275.

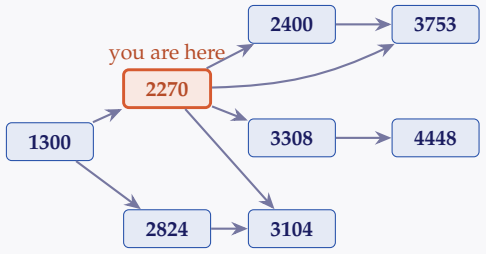
CSCI 3104 Algorithms
Requires (CSCI 2270 or 2275) and (APPM 1360 or MATH 2300) and (CSCI 2824 or …).

CSCI 3753 Operating Systems
Requires (CSCI 2270 or 2275) and (CSCI 2400 or …).

CU Boulder catalog 2026–27, shortened

model it

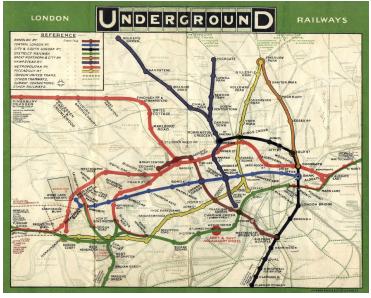
The graph



No cycles allowed. A cycle here would mean nobody could ever graduate.

Checklist V = courses | E = "required before" | Directed? yes | Weighted? no | Sparse or dense?

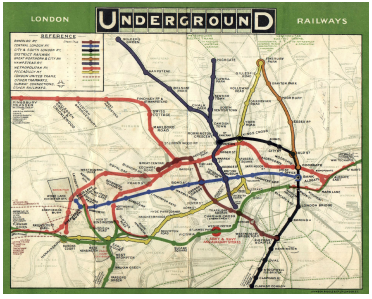
The most famous graph drawing in the world



Before: drawn to scale (1908)

Lines drawn on top of the street map. The middle is so crowded that the outer ends of two lines were simply left off.

The most famous graph drawing in the world



Before: drawn to scale (1908)

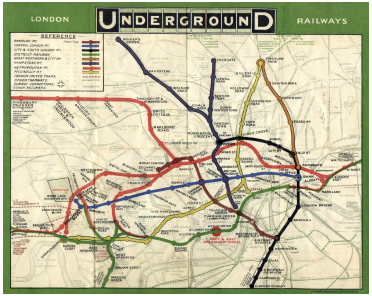
Lines drawn on top of the street map. The middle is so crowded that the outer ends of two lines were simply left off.



Harry Beck, 1931

Underground, riders only need the order of the stations and where to change. So he threw the geography away: straight lines, 45° angles, even spacing.

The most famous graph drawing in the world



Before: drawn to scale (1908)

Lines drawn on top of the street map. The middle is so crowded that the outer ends of two lines were simply left off.

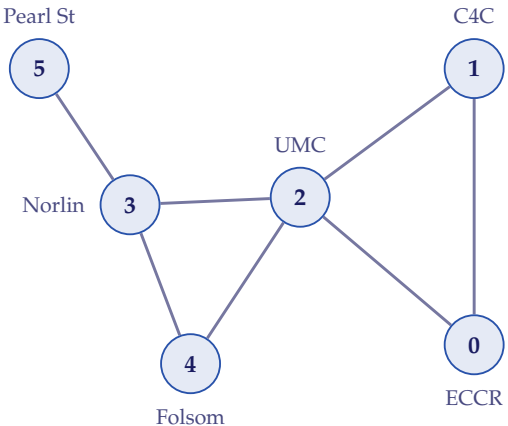


Harry Beck, 1931

Underground, riders only need the order of the stations and where to change. So he threw the geography away: straight lines, 45° angles, even spacing.

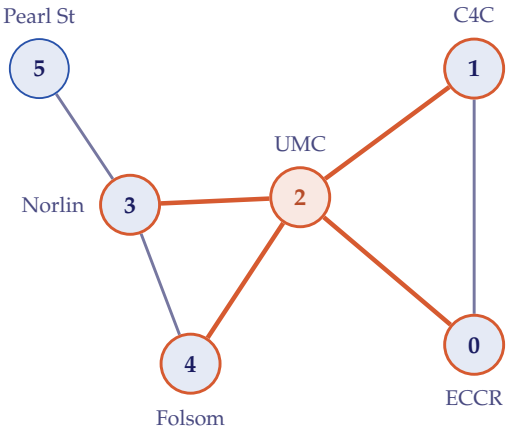
Same move as Euler: keep only what's connected to what.

The words, all on one graph



A graph $G = (V, E)$
Vertices $V = \{0, \ldots, 5\}$, edges E: 7 pairs $\{u, v\}$.
$n = |V| = 6$, $m = |E| = 7$.

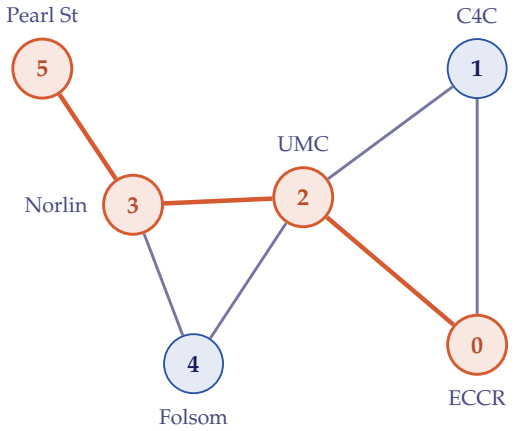
The words, all on one graph



A graph $G = (V, E)$
Vertices $V = \{0, \ldots, 5\}$, edges E: 7 pairs $\{u, v\}$.
$n = |V| = 6$, $m = |E| = 7$.

Neighbors and degree
UMC's neighbors: 0, 1, 3, 4, so $\deg(\text{UMC}) = 4$.
Directed?: in-degree and out-degree.

The words, all on one graph

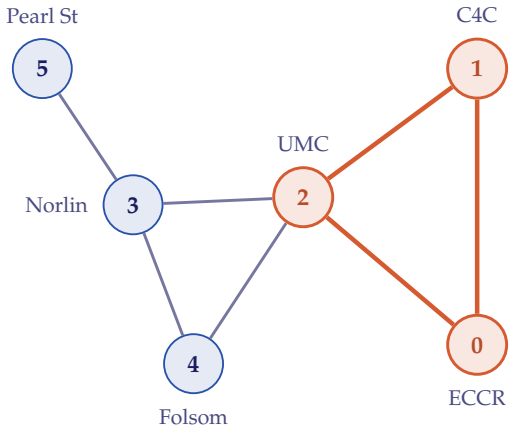


A graph $G = (V, E)$
Vertices $V = \{0, \ldots, 5\}$, edges E: 7 pairs $\{u, v\}$.
$n = |V| = 6$, $m = |E| = 7$.

Neighbors and degree
UMC's neighbors: 0, 1, 3, 4, so $\deg(\text{UMC}) = 4$.
Directed?: in-degree and out-degree.

Path
Pearl $\to$ Norlin $\to$ UMC $\to$ ECCR.
Length 3: count the edges.

The words, all on one graph



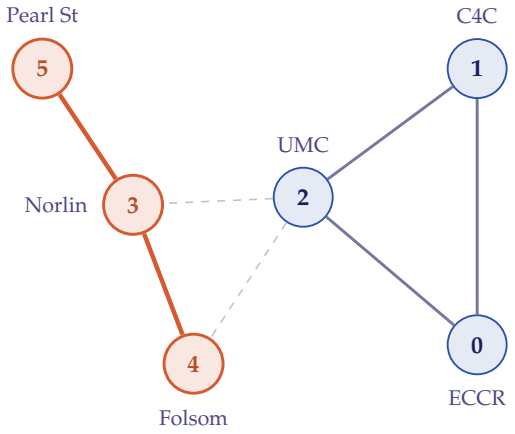
A graph $G = (V, E)$
Vertices $V = \{0, \ldots, 5\}$, edges E: 7 pairs $\{u, v\}$.
$n = |V| = 6$, $m = |E| = 7$.

Neighbors and degree
UMC's neighbors: 0, 1, 3, 4, so $\deg(\text{UMC}) = 4$.
Directed?: in-degree and out-degree.

Path
Pearl $\to$ Norlin $\to$ UMC $\to$ ECCR.
Length 3: count the edges.

Cycle
UMC $\to$ C4C $\to$ ECCR $\to$ UMC.
A path that ends where it started.

The words, all on one graph



A graph $G = (V, E)$
Vertices $V = \{0, \ldots, 5\}$, edges E: 7 pairs $\{u, v\}$.
$n = |V| = 6$, $m = |E| = 7$.

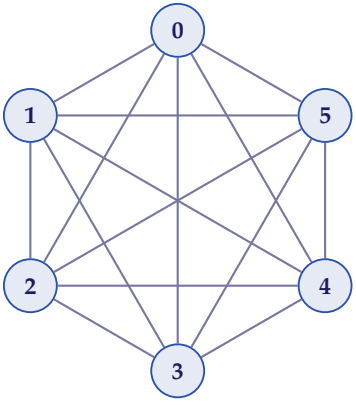
Neighbors and degree
UMC's neighbors: 0, 1, 3, 4, so $\deg(\text{UMC}) = 4$.
Directed?: in-degree and out-degree.

Path
Pearl $\to$ Norlin $\to$ UMC $\to$ ECCR.
Length 3: count the edges.

Cycle
UMC $\to$ C4C $\to$ ECCR $\to$ UMC.
A path that ends where it started.

Connected
A path between every pair. Delete the two dashed edges and it falls into two pieces.

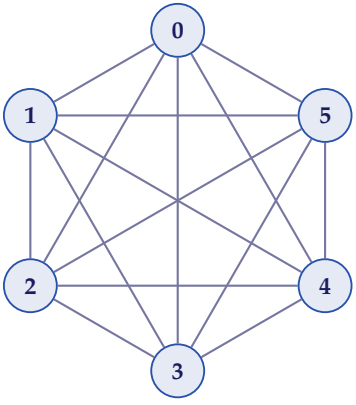
How many edges can a graph have?



The question

With n vertices, what is the **most** edges m can be?

How many edges can a graph have?



K_6: every pair joined, 15 edges

The question

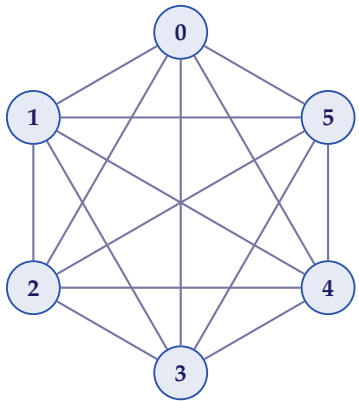
With n vertices, what is the **most** edges m can be?

Count the ends

Each vertex can join $n-1$ others: $n(n-1)$ ends. Each edge has two ends, so

$$m \leq \frac{n(n-1)}{2}$$

How many edges can a graph have?



K_6: every pair joined, 15 edges

The question

With n vertices, what is the **most** edges m can be?

Count the ends

Each vertex can join $n-1$ others: $n(n-1)$ ends. Each edge has two ends, so

$$m \leq \frac{n(n-1)}{2}$$

Real graphs are far from full

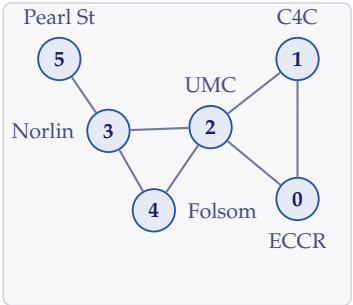
Campus: 7 of 15 possible. Facebook: about 4.5×10^{11} of 4.5×10^{18}.

Sparse: m is far below n^2. **Dense**: m is close to $n^2/2$.

Two ways to store the same graph

Must answer fast: **(1)** Is u–v an edge? **(2)** Who are u's neighbors?

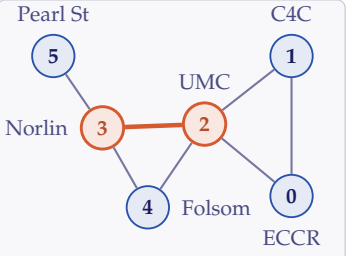
The graph $n = 6$, $m = 7$



Two ways to store the same graph

Must answer fast: **(1)** Is u–v an edge? **(2)** Who are u's neighbors?

The graph $n = 6,\ m = 7$



one edge: {UMC, Norlin} = {2, 3}

Adjacency matrix A

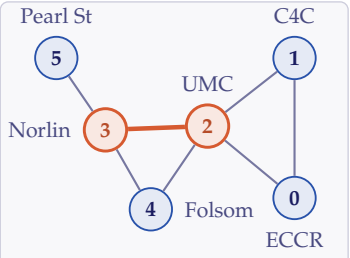
	0	1	2	3	4	5
0	0	**1**	**1**	0	0	0
1	**1**	0	**1**	0	0	0
2	**1**	**1**	0	**1**	**1**	0
3	0	0	**1**	0	**1**	**1**
4	0	0	**1**	**1**	0	0
5	0	0	0	**1**	0	0

edge $\{u, v\}$: two cells
$A[u][v] = A[v][u] = 1$

Two ways to store the same graph

Must answer fast: **(1)** Is u–v an edge? **(2)** Who are u's neighbors?

The graph $n = 6,\ m = 7$



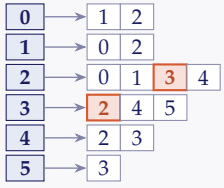
one edge: {UMC, Norlin} = {2, 3}

Adjacency matrix A

	0	1	2	3	4	5
0	0	1	1	0	0	0
1	1	0	1	0	0	0
2	1	1	0	1	1	0
3	0	0	1	0	1	1
4	0	0	1	1	0	0
5	0	0	0	1	0	0

edge $\{u, v\}$: two cells
$A[u][v] = A[v][u] = 1$

Adjacency list (an array of vectors)

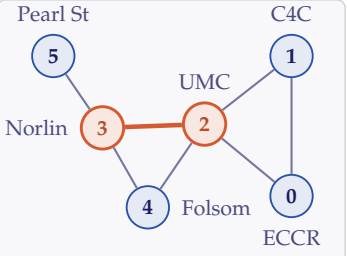


edge $\{u, v\}$: two entries
v in u's list, u in v's list

Two ways to store the same graph

Must answer fast: **(1)** Is u–v an edge? **(2)** Who are u's neighbors?

The graph $n = 6,\ m = 7$



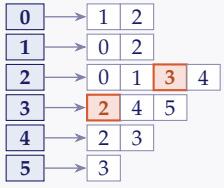
one edge: {UMC, Norlin} = {2, 3}

Adjacency matrix A

	0	1	2	3	4	5
0	0	1	1	0	0	0
1	1	0	1	0	0	0
2	1	1	0	1	1	0
3	0	0	1	0	1	1
4	0	0	1	1	0	0
5	0	0	0	1	0	0

edge $\{u, v\}$: two cells
$A[u][v] = A[v][u] = 1$

Adjacency list (an array of vectors)



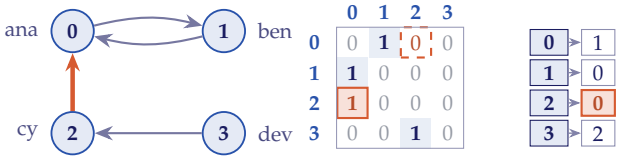
edge $\{u, v\}$: two entries
v in u's list, u in v's list

Same 7 edges: matrix stores **36 cells** (22 are zeros) · list stores **14 entries**

Now picture 3 billion vertices…

Directed? Weighted? Two small changes

Directed: the follow graph

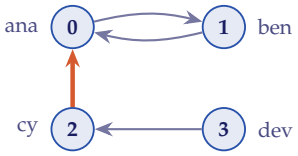


Edge $u \to v$ sets only $A[u][v]$, so the matrix is **no longer symmetric**: cy follows ana, ana doesn't follow cy.

Only u's list gets v.

Directed? Weighted? Two small changes

Directed: the follow graph



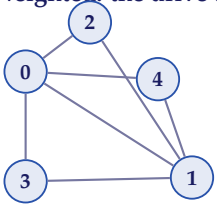
	0	1	2	3
0	0	1	0	0
1	1	0	0	0
2	1	0	0	0
3	0	0	1	0

0	1
1	0
2	0
3	2

Edge $u \to v$ sets only $A[u][v]$, so the matrix is **no longer symmetric**: cy follows ana, ana doesn't follow cy.

Only u's list gets v.

Weighted: the drive times



	0	1	2	3	4
0	·	35	25	30	15
1	35	·	40	25	25
2	25	40	·	·	·
3	30	25	·	·	·
4	15	25	·	·	·

0	1:35	2:25	3:30	4:15
1	0:35	2:40	3:25	4:25
2	0:25	1:40		
3	0:30	1:25		
4	0:15	1:25		

Store the **weight** instead of a 1, and use a marker for "no edge".

List entries become pairs: (neighbor, weight). Written here as `to:w`.

In C++

```
struct Edge { int to, weight; };
class Graph {
    vector<vector<Edge» adj;           // adj[u] = u's edges
public:
    Graph(int n) :  adj(n) {}
    void addEdge(int u, int v, int w = 1) {
        adj[u].push_back({v, w});
        adj[v].push_back({u, w});      // skip if directed
    }
    bool hasEdge(int u, int v) {
        for (const Edge& e :  adj[u]) // O(deg u)
            if (e.to == v) return true;
        return false;
    }
    vector<Edge>& neighbors(int u) {
        return adj[u];                 // O(1)
    }
};
```

The data

`adj[u]` is a vector of u's edges, each a (to, weight) pair. Exactly the list picture.

In C++

```cpp
struct Edge { int to, weight; };
class Graph {
    vector<vector<Edge>> adj;          // adj[u] = u's edges
public:
    Graph(int n) :  adj(n) {}
    void addEdge(int u, int v, int w = 1) {
        adj[u].push_back({v, w});
        adj[v].push_back({u, w});      // skip if directed
    }
    bool hasEdge(int u, int v) {
        for (const Edge& e :  adj[u]) // O(deg u)
            if (e.to == v) return true;
        return false;
    }
    vector<Edge>& neighbors(int u) {
        return adj[u];                 // O(1)
    }
};
```

The data

`adj[u]` is a vector of u's edges, each a (to, weight) pair. Exactly the list picture.

Adding an edge: $O(1)$

Undirected means two pushes, one in each direction.

In C++

```cpp
struct Edge { int to, weight; };
class Graph {
    vector<vector<Edge>> adj;          // adj[u] = u's edges
public:
    Graph(int n) :  adj(n) {}
    void addEdge(int u, int v, int w = 1) {
        adj[u].push_back({v, w});
        adj[v].push_back({u, w});      // skip if directed
    }
    bool hasEdge(int u, int v) {
        for (const Edge& e :  adj[u]) // O(deg u)
            if (e.to == v) return true;
        return false;
    }
    vector<Edge>& neighbors(int u) {
        return adj[u];                 // O(1)
    }
};
```

The data

`adj[u]` is a vector of u's edges, each a (to, weight) pair. Exactly the list picture.

Adding an edge: $O(1)$

Undirected means two pushes, one in each direction.

Queries

`hasEdge` scans one list: $O(\deg u)$. `neighbors` just hands the list back.

In C++

```
struct Edge { int to, weight; };
class Graph {
    vector<vector<Edge>> adj;          // adj[u] = u's edges
public:
    Graph(int n) :  adj(n) {}
    void addEdge(int u, int v, int w = 1) {
        adj[u].push_back({v, w});
        adj[v].push_back({u, w});      // skip if directed
    }
    bool hasEdge(int u, int v) {
        for (const Edge& e :  adj[u]) // O(deg u)
            if (e.to == v) return true;
        return false;
    }
    vector<Edge>& neighbors(int u) {
        return adj[u];                 // O(1)
    }
};
```

The data

`adj[u]` is a vector of u's edges, each a (to, weight) pair. Exactly the list picture.

Adding an edge: $O(1)$

Undirected means two pushes, one in each direction.

Queries

`hasEdge` scans one list: $O(\deg u)$. `neighbors` just hands the list back.

The matrix version, for comparison

```
vector<vector<int>> A(n, vector<int>(n));
A[u][v] = A[v][u] = w;     // O(1)
bool has = (A[u][v] != 0); // O(1)
```

What about 3 billion vertices?

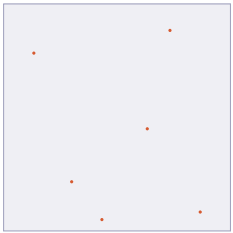
Facebook: about $n = 3 \times 10^9$ users, each with a few hundred friends. Which would you build?

Checklist V = users | E = friendships | Directed? no | Weighted? no | Sparse or dense?

What about 3 billion vertices?

Facebook: about $n = 3 \times 10^9$ users, each with a few hundred friends. Which would you build?

Adjacency matrix



the coral specks are the 1s

$n^2 = 9 \times 10^{18}$ cells. At 1 bit each: about **1.1 exabytes**.

Checklist V = users | E = friendships | Directed? no | Weighted? no | Sparse or dense?

What about 3 billion vertices?

Facebook: about $n = 3 \times 10^9$ users, each with a few hundred friends. Which would you build?

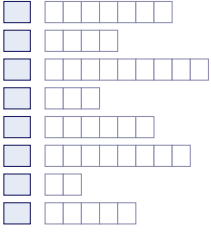
Adjacency matrix



the coral specks are the 1s

$n^2 = 9 \times 10^{18}$ cells. At 1 bit each: about **1.1 exabytes**.

Adjacency list



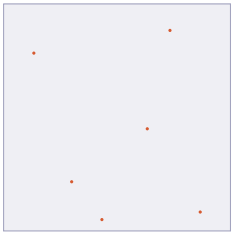
$n + 2m \approx 10^{12}$ entries. At 4 bytes each: a few **terabytes**, about 300,000× smaller.

Checklist V = users | E = friendships | Directed? no | Weighted? no | Sparse or dense?

What about 3 billion vertices?

Facebook: about $n = 3 \times 10^9$ users, each with a few hundred friends. Which would you build?

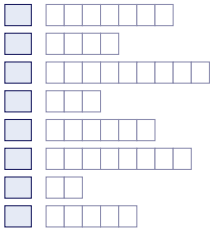
Adjacency matrix



the coral specks are the 1s

$n^2 = 9 \times 10^{18}$ cells. At 1 bit each: about **1.1 exabytes**.

Adjacency list



$n + 2m \approx 10^{12}$ entries. At 4 bytes each: a few **terabytes**, about 300,000× smaller.



Only 1 cell in 10 million is a 1. This graph is **sparse**, and real graphs almost always are.

Checklist V = users | E = friendships | Directed? no | Weighted? no | Sparse → list

The trade-off, in one table

Operation	Matrix	List
Space	$O(n^2)$	$O(n+m)$

The trade-off, in one table

Operation	Matrix	List
Space	$O(n^2)$	$O(n+m)$
Is u–v an edge?	$O(1)$	$O(\deg u)$
List u's neighbors	$O(n)$	$O(\deg u)$

The trade-off, in one table

Operation	Matrix	List
Space	$O(n^2)$	$O(n+m)$
Is u–v an edge?	$O(1)$	$O(\deg u)$
List u's neighbors	$O(n)$	$O(\deg u)$
Add an edge	$O(1)$	$O(1)$

Rule of thumb. Most real graphs are sparse, so the **adjacency list** is the default. Use a **matrix** when the graph is dense, or when you need $O(1)$ edge checks.

Hands up: matrix or list?

Colorado road map

Intersections, and the road segments between them.

30 airport hubs

Flights between almost every pair of hubs.

Wikipedia

About 7 million articles, and the links between them.

Hands up: matrix or list?

Colorado road map

Intersections, and the road segments between them.

List

Each intersection meets about 4 roads, out of millions: very sparse.

30 airport hubs

Flights between almost every pair of hubs.

Wikipedia

About 7 million articles, and the links between them.

Hands up: matrix or list?

Colorado road map

Intersections, and the road segments between them.

List

Each intersection meets about 4 roads, out of millions: very sparse.

30 airport hubs

Flights between almost every pair of hubs.

Matrix

Dense, and only $30 \times 30 = 900$ cells. $O(1)$ lookups.

Wikipedia

About 7 million articles, and the links between them.

Hands up: matrix or list?

Colorado road map

Intersections, and the road segments between them.

List

Each intersection meets about 4 roads, out of millions: very sparse.

30 airport hubs

Flights between almost every pair of hubs.

Matrix

Dense, and only $30 \times 30 = 900$ cells. $O(1)$ lookups.

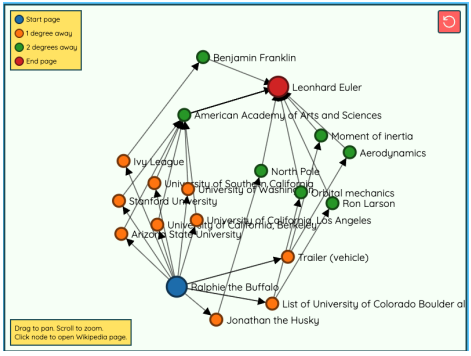
Wikipedia

About 7 million articles, and the links between them.

List

A few dozen links per article, out of millions.

Six Degrees of Wikipedia



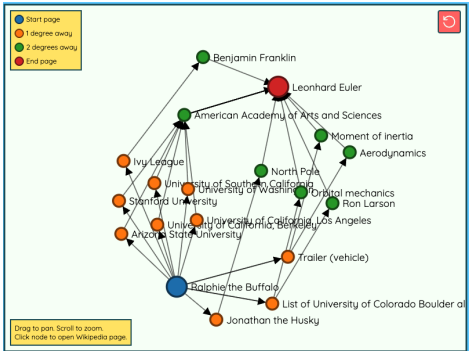
sixdegreesofwikipedia.com

Wikipedia is a graph

Run the checklist on it. What are V and E? Directed? Weighted? Sparse?

Checklist | V = ? | E = ? | Directed? | Weighted? | Sparse or dense?

Six Degrees of Wikipedia



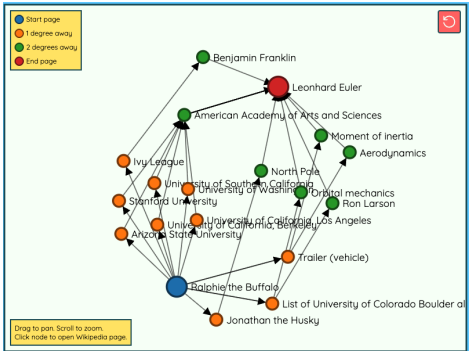
sixdegreesofwikipedia.com

Wikipedia is a graph

Run the checklist on it. What are V and E? Directed? Weighted? Sparse?

Checklist V = articles | E = links | Directed? yes | Weighted? no | Sparse → list

Six Degrees of Wikipedia



sixdegreesofwikipedia.com

Wikipedia is a graph

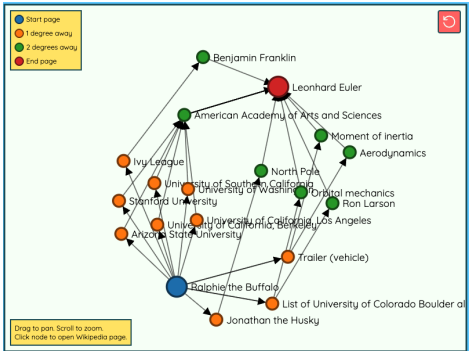
Run the checklist on it. What are V and E? Directed? Weighted? Sparse?

Live demo

From **Ralphie** to a page you pick. The site finds the fewest clicks between them.

Checklist V = articles | E = links | Directed? yes | Weighted? no | Sparse → list

Six Degrees of Wikipedia



sixdegreesofwikipedia.com

Wikipedia is a graph

Run the checklist on it. What are V and E? Directed? Weighted? Sparse?

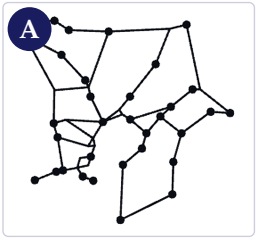
Live demo

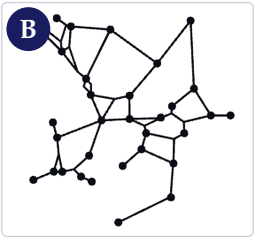
From **Ralphie** to a page you pick. The site finds the fewest clicks between them.

It searched about 7 million vertices almost instantly. **How?** That's the next lecture.

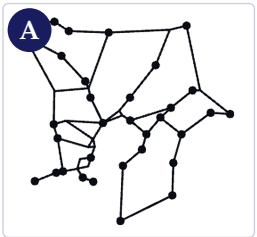
Checklist | V = articles | E = links | Directed? yes | Weighted? no | Sparse → list

Remember A and B?

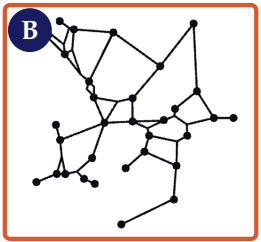




Remember A and B?



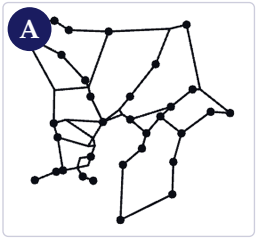
Tokyo rail, decades



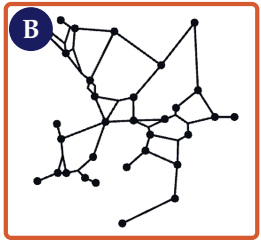
slime mold, about 1 day

In about a day it grew a network with cost, efficiency and fault tolerance **comparable to the real one**.

Remember A and B?



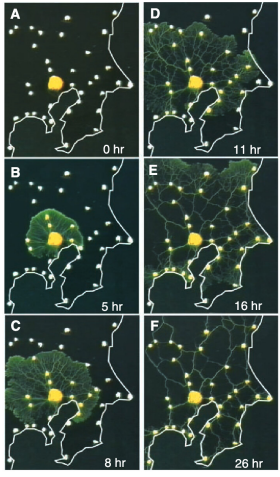
Tokyo rail, decades



slime mold, about 1 day

In about a day it grew a network with cost, efficiency and fault tolerance **comparable to the real one**.

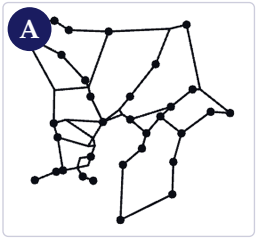
One cell, no brain. Oat flakes marked the cities; light (which it avoids) marked mountains and sea. It spread everywhere, then kept only the tubes that carried food.



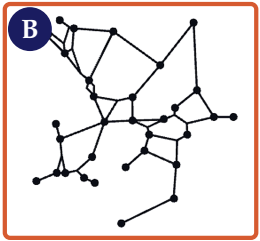


Time-lapse of the experiment: youtu.be/BZUQQmcR5-g

Remember A and B?



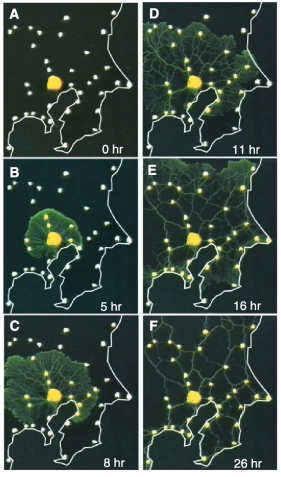
Tokyo rail, decades



slime mold, about 1 day

In about a day it grew a network with cost, efficiency and fault tolerance **comparable to the real one**.

One cell, no brain. Oat flakes marked the cities; light (which it avoids) marked mountains and sea. It spread everywhere, then kept only the tubes that carried food.





Time-lapse of the experiment: youtu.be/BZUQQmcR5-g

Same lab, earlier: it finds the shortest path through a maze.

Two Ig Nobel Prizes (2008, 2010).

Three things to keep

1 A graph keeps only the connections.
$G = (V, E)$: vertices, and edges between them. Euler, Beck and the slime mold all made the same move.

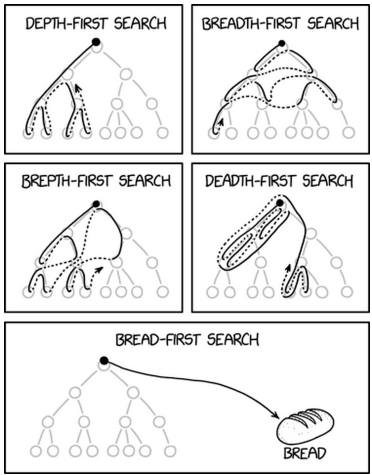
2 Ask the checklist.
What are V and E? Directed? Weighted? Sparse or dense?

3 Pick the structure.
Adjacency list for sparse graphs (most real ones). Matrix for dense graphs or $O(1)$ edge checks.



Your checklist V = ? | E = ? | Directed? | Weighted? | Sparse or dense? → list or matrix

Next time



xkcd.com/2407 by Randall Munroe, CC BY-NC 2.5

Next lecture

How do you **search** a graph?

Two ways to explore from a starting vertex:

Depth-first: go as deep as you can, then back up.

Breadth-first: visit all your neighbors first, then theirs.

Six Degrees runs breadth-first search from both ends at once to find Ralphie's shortest paths.

Bread-first search will not be on the exam.