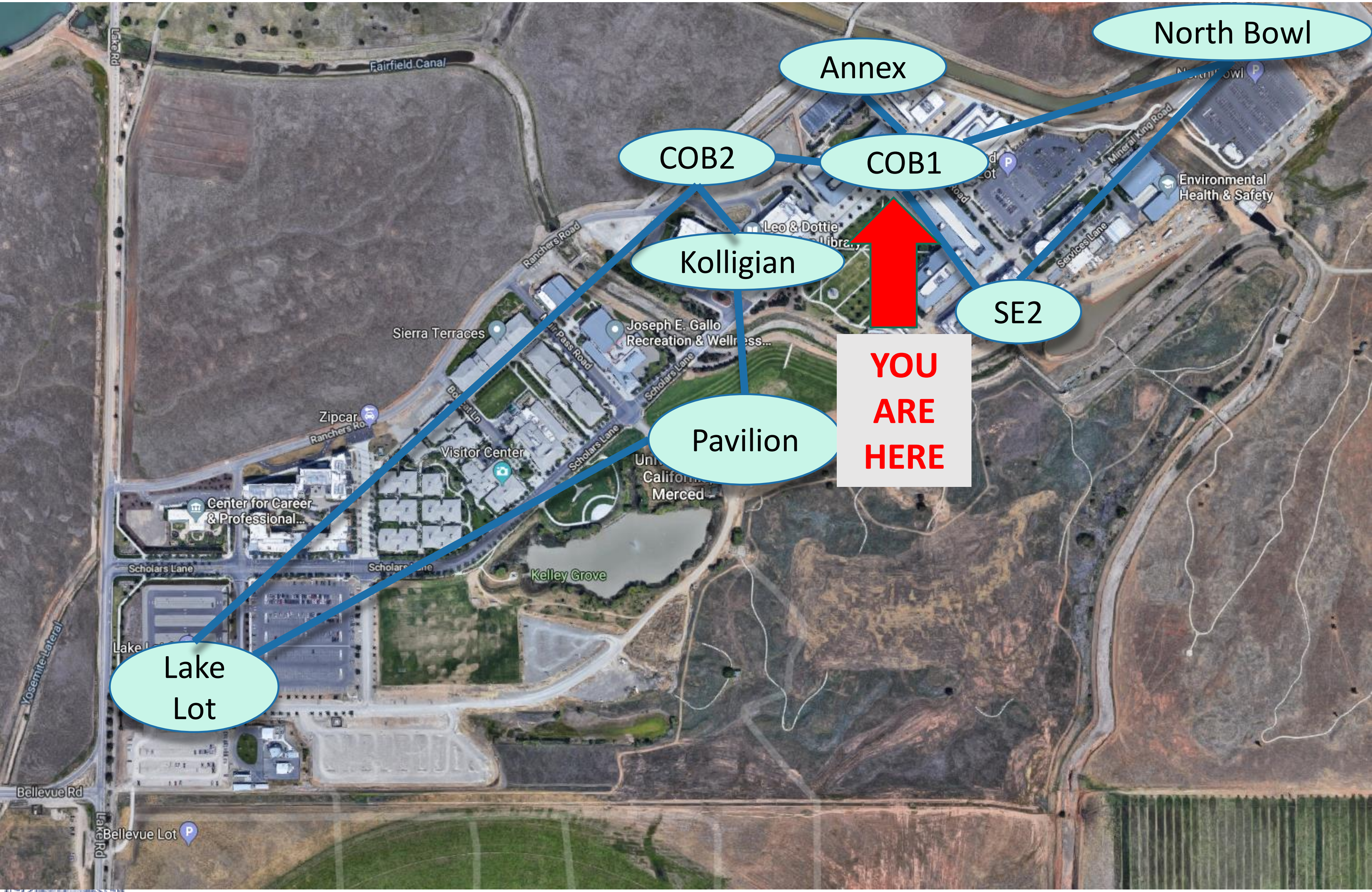
CSE100: Design and Analysis of Algorithms
Lecture 19 – Weighted Graphs

Apr 5th 2022

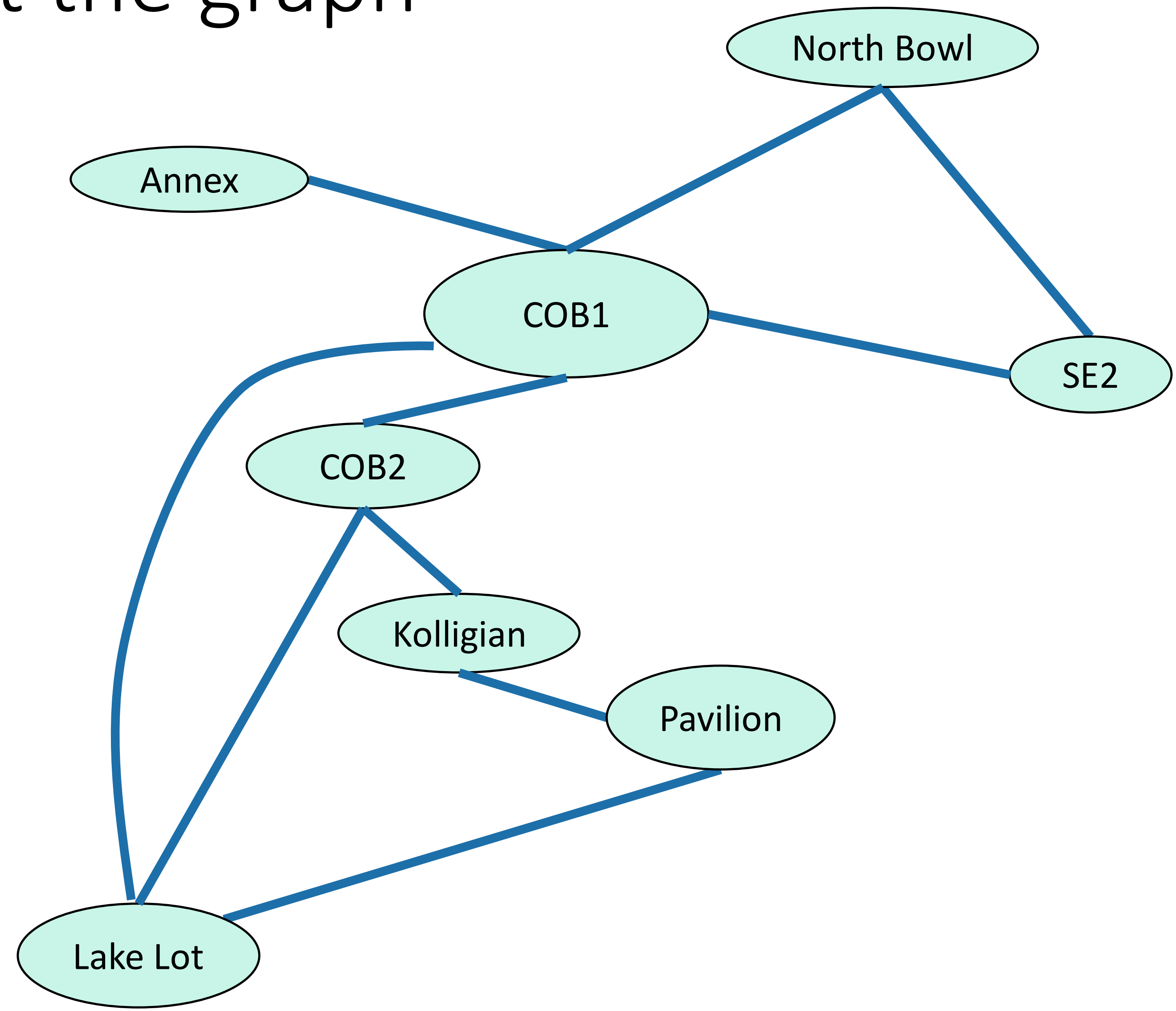
Dijkstra and Bellman-Ford

Today

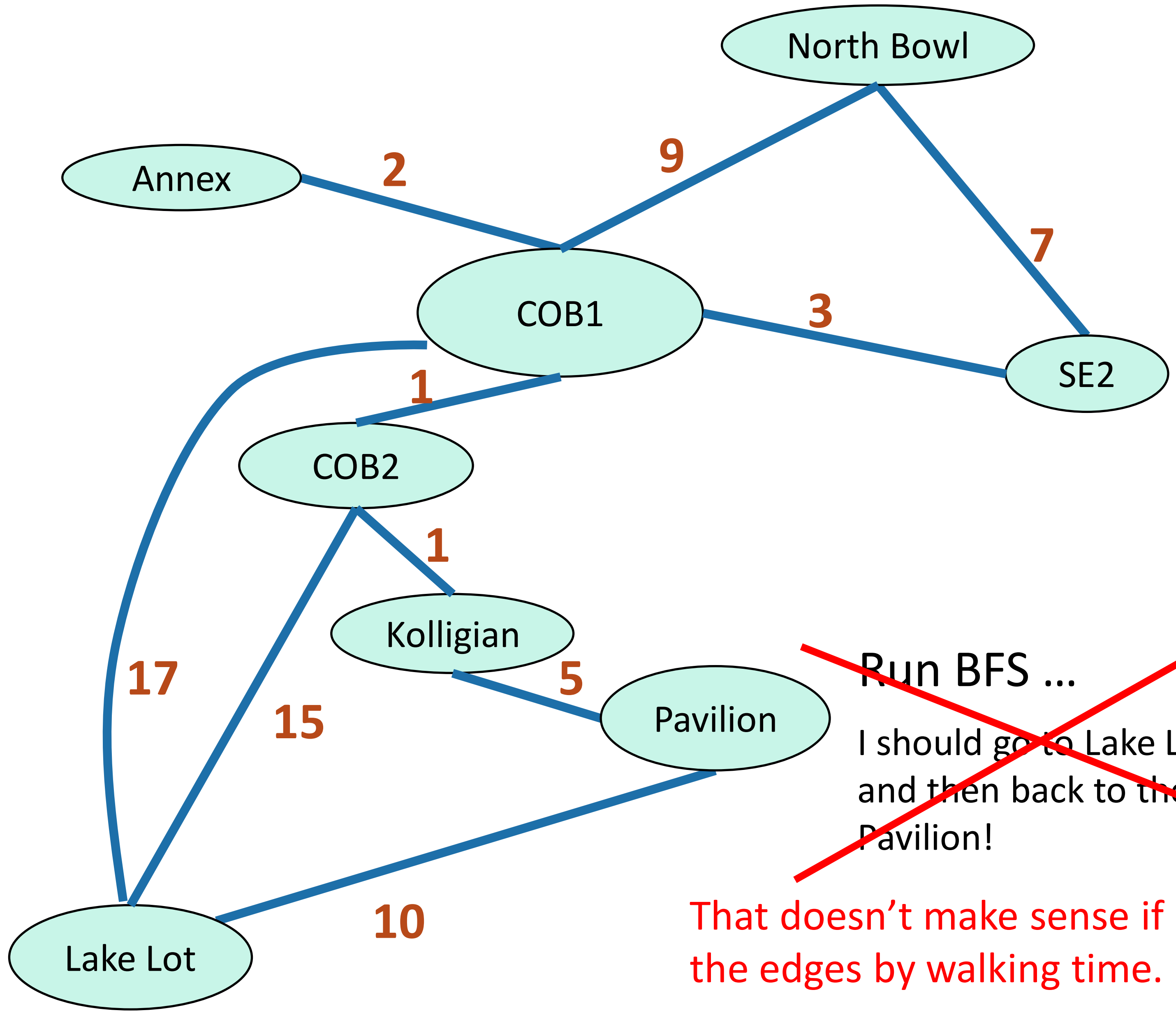
- What if the graphs are weighted?
- Part A: Dijkstra!
  - This will take most of today's class
- Part B: Bellman-Ford!
  - Real quick at the end!
  - We'll come back to Bellman-Ford in more detail, so today is just a taste.


North Bowl
Annex
COB2
COB1
Kolligian
SE2
Pavilion
Lake Lot
YOU ARE HERE
Fairfield Canal
Lake Rd
Ranchers Road
Mineral King Road
Environmental Health & Safety
Services Lane
Sierra Terraces
Joseph E. Gallo Recreation & Wellness...
Zipcar
Visitor Center
Center for Career & Professional...
Scholars Lane
Kelley Grove
Yosemite Lateral
Bellevue Rd
Bellevue Lot

Just the graph



Shortest path from COB1 to Pavilion?

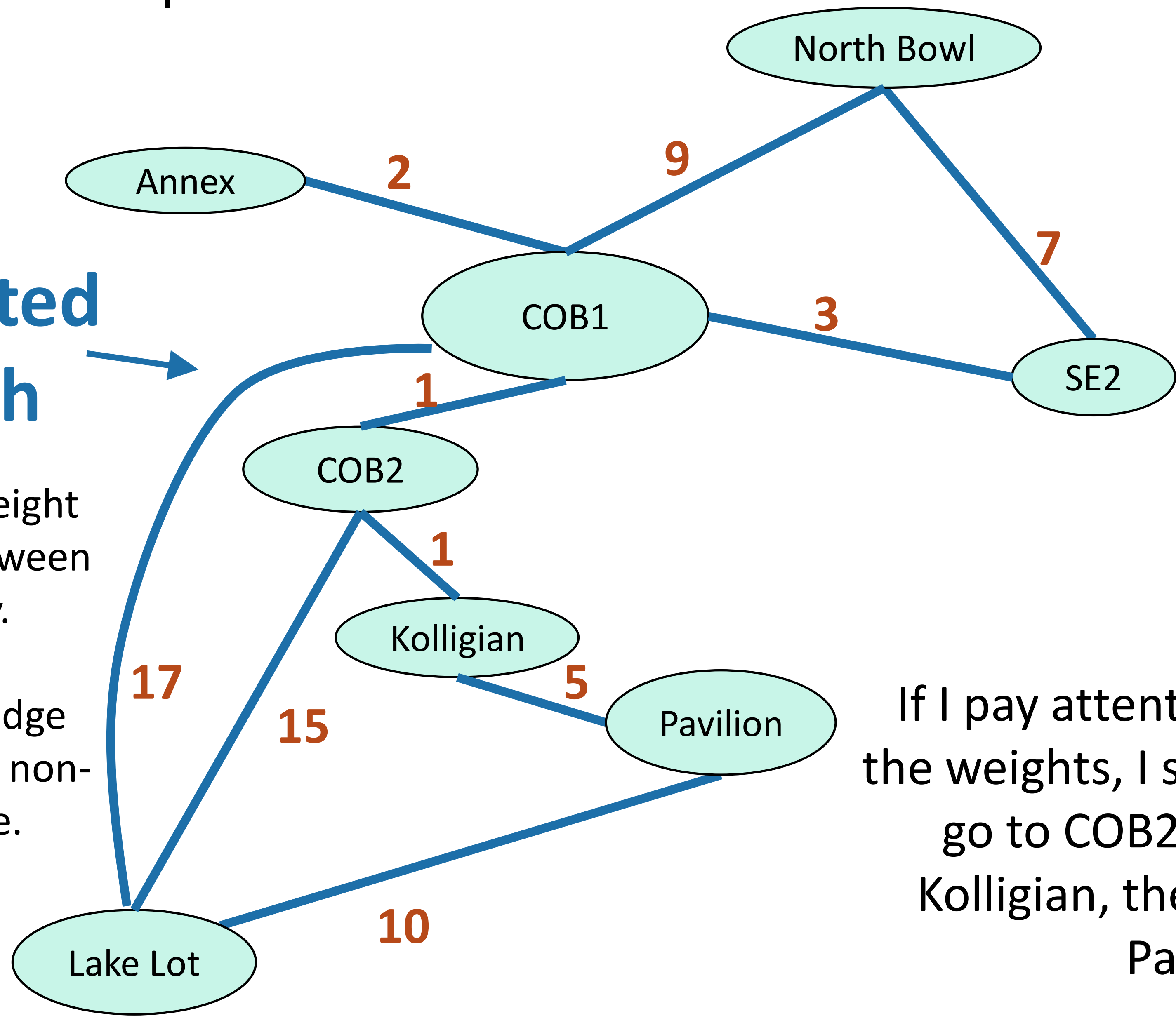


~~Run BFS …~~

~~I should go to Lake Lot and then back to the Pavilion!~~

That doesn't make sense if I label the edges by walking time.

Shortest path from COB1 to the Pavilion?



weighted graph

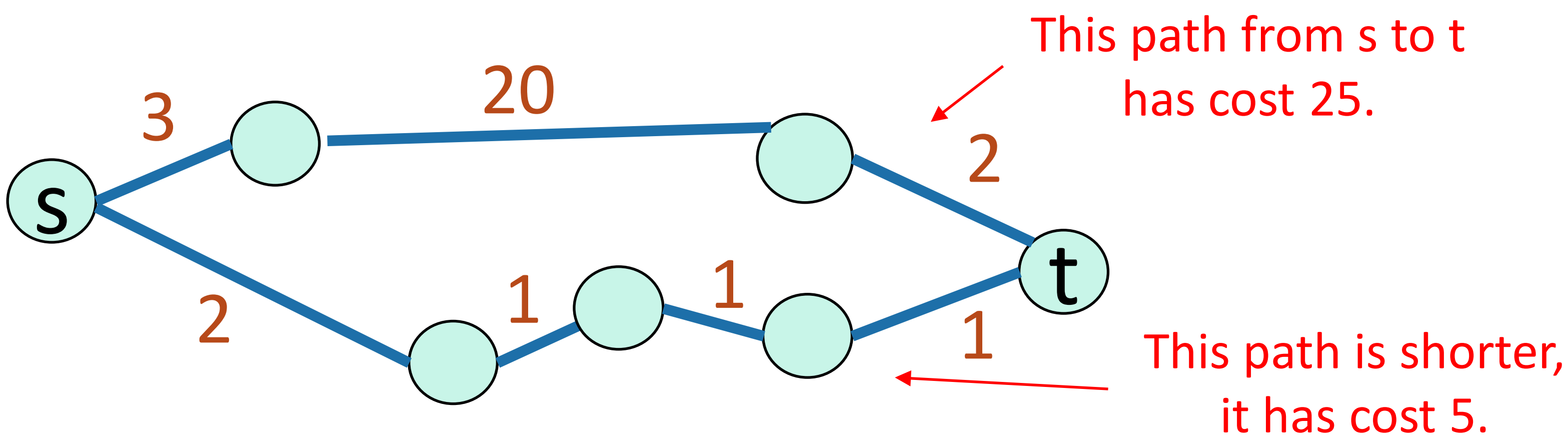
w(u,v) = weight of edge between u and v.

For now, edge weights are non-negative.

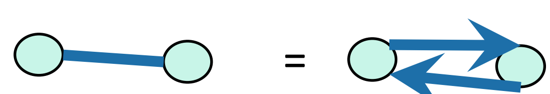
If I pay attention to the weights, I should go to COB2, then Kolligian, then the Pavilion.

Shortest path problem

- What is the shortest path between u and v in a weighted graph?
  - the **cost** of a path is the sum of the weights along that path
  - The **shortest path** is the one with the minimum cost.



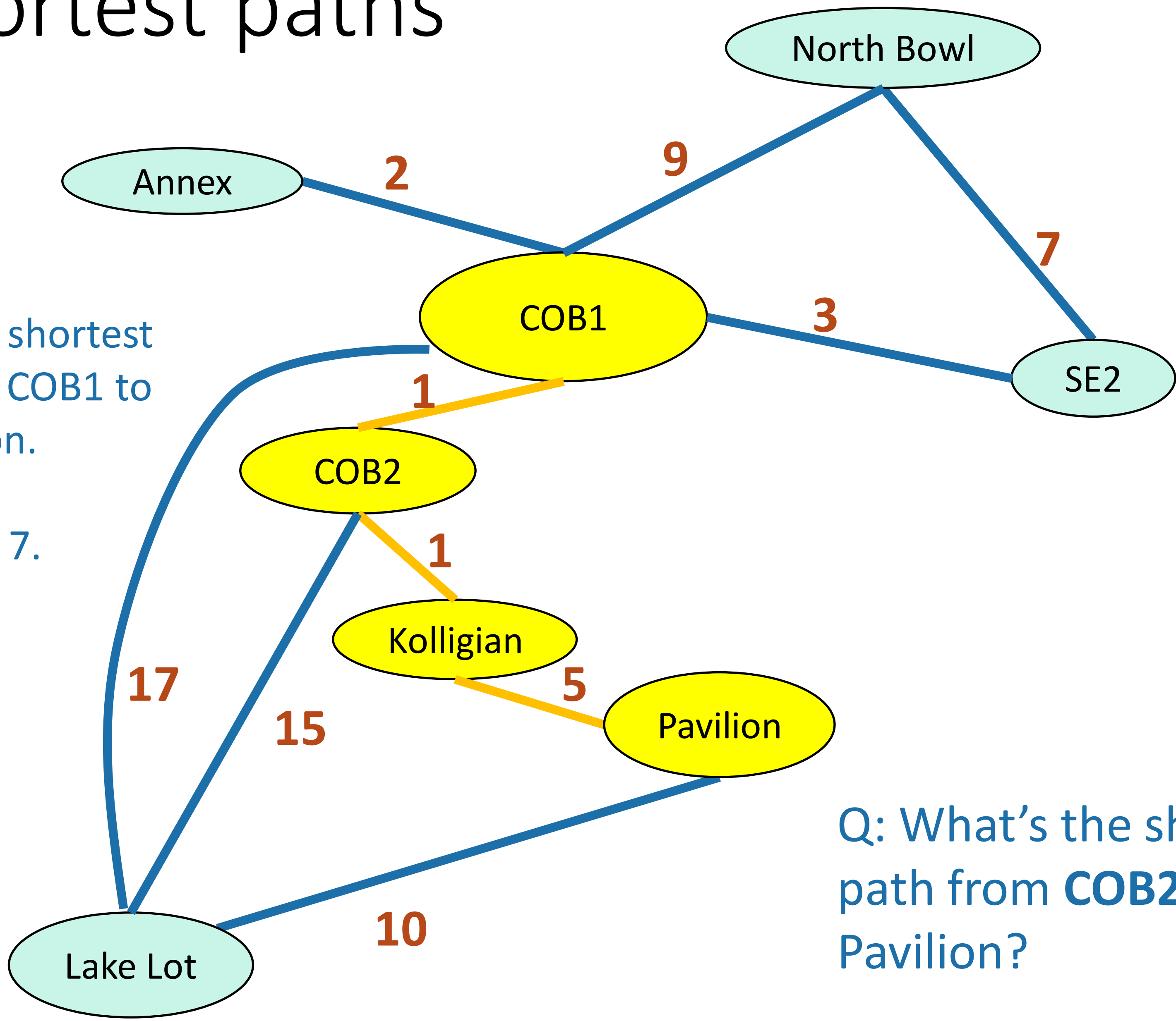
- The **distance** d(u,v) between two vertices u and v is the cost of the shortest path between u and v.
- For this lecture **all graphs are directed**, but to save on notation I'm just going to draw undirected edges.



Shortest paths

This is the shortest path from COB1 to the Pavilion.

It has cost 7.

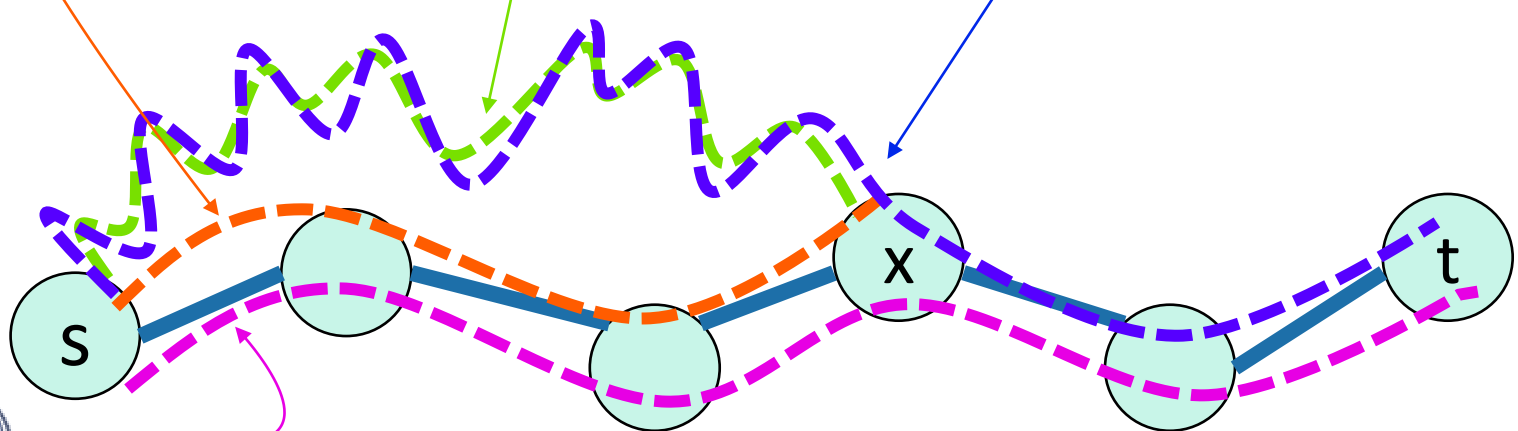


Q: What's the shortest path from **COB2** to the Pavilion?

Warm-up

- A sub-path of a shortest path is also a shortest path.

- Say **this** is a shortest path from s to t.
- Claim: **this** is a shortest path from s to x.
  - Suppose not, **this** one is a shorter path from s to x.
  - But then that gives an **even shorter path** from s to t!



Single-source shortest-path problem

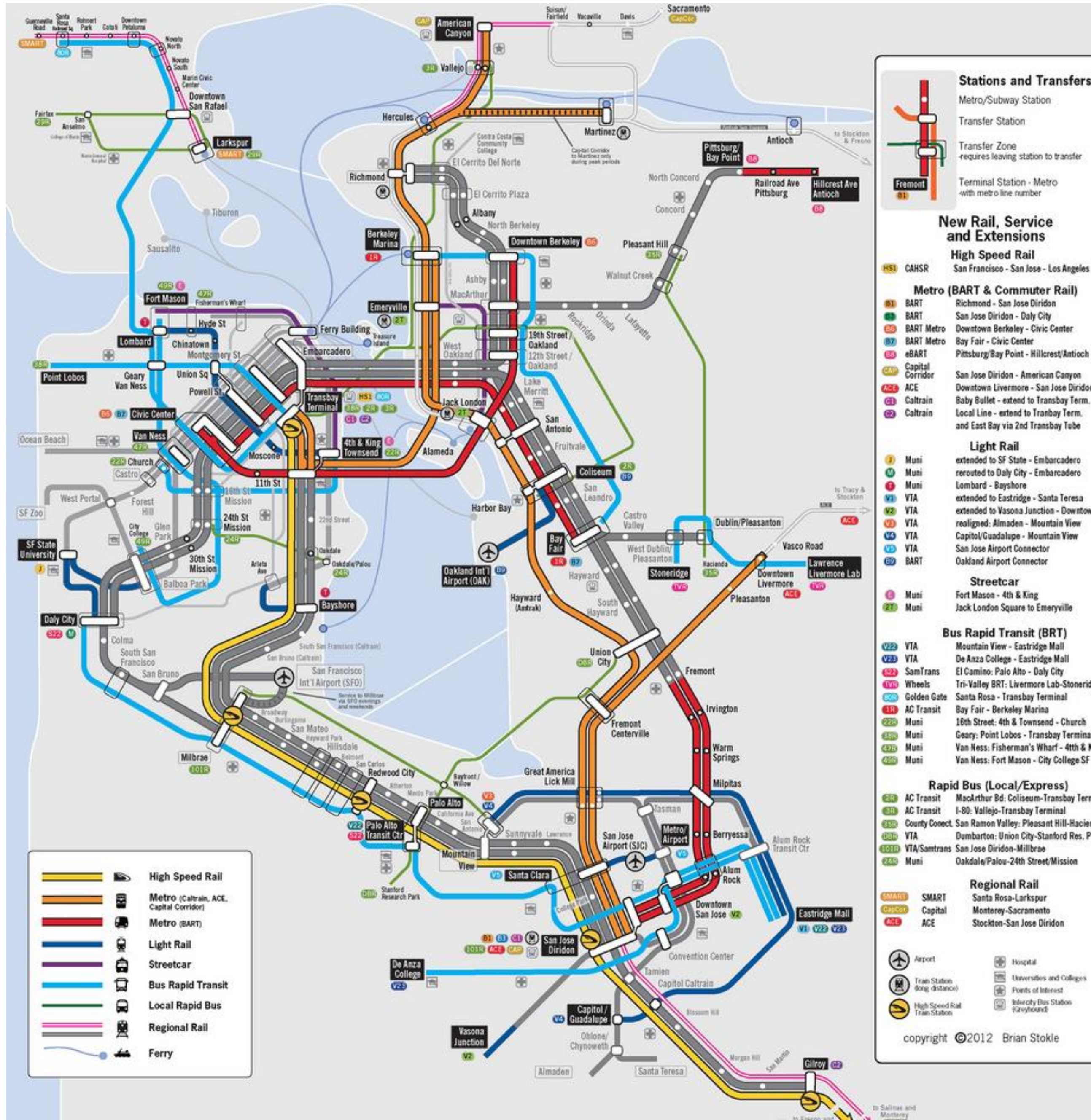
- I want to know the shortest path from one vertex (COB1) to all other vertices.

Destination	Cost	To get there
COB2	1	COB2
Kolligian	2	COB2-Kolligian
Annex	2	Annex
North Bowl	9	North Bowl
Pavilion	7	COB2-Kolligian-Pavilion
SE2	3	SE2
Lake Lot	16	COB2-Lake Lot

(Not necessarily stored as a table – how this information is represented will depend on the application)

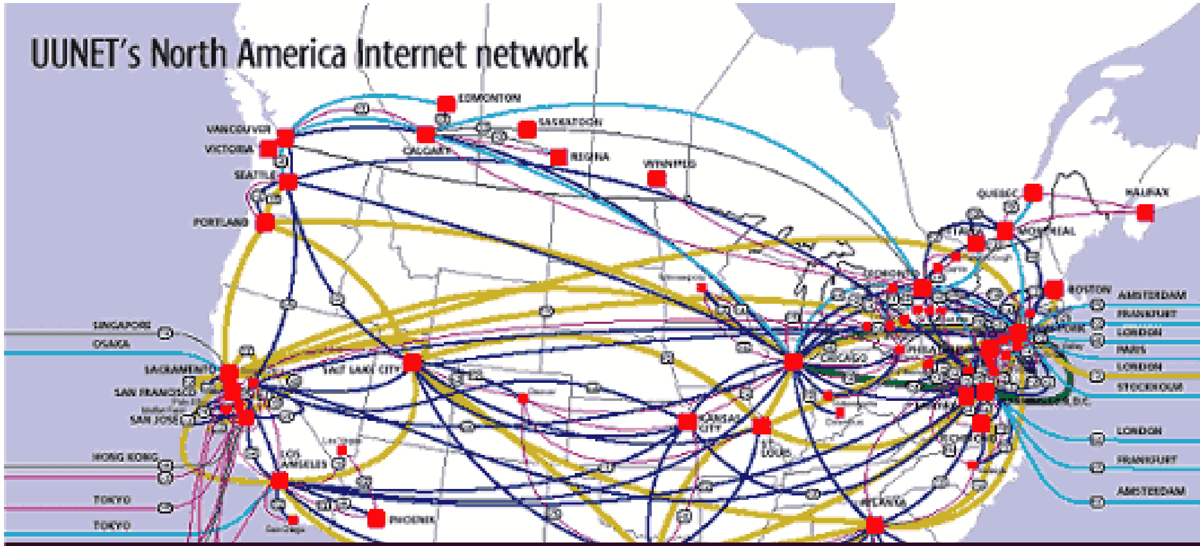
Example

- **“what is the shortest path from Palo Alto to [anywhere else]”** using BART, Caltrain, lightrail, MUNI, bus, Amtrak, bike, walking, uber/lyft.
- Edge weights have something to do with time, money, hassle.



Example

- **Network routing**
- I send information over the internet, from my computer to to all over the world.
- Each path has a cost which depends on link length, traffic, other costs, etc..
- How should we send packets?



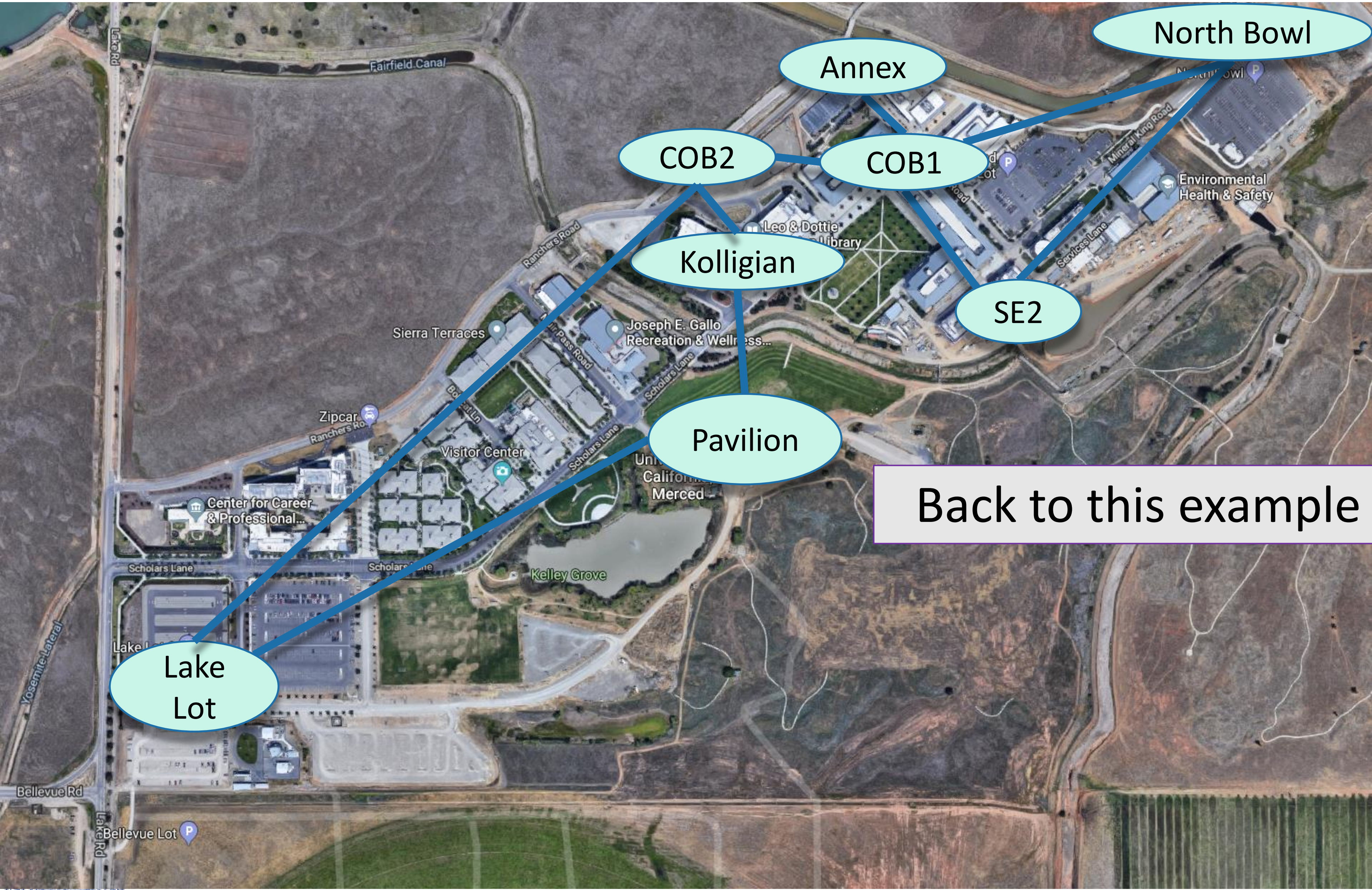

```
7:33pm root@mars:~#[5]traceroute -A www.ethz.ch
traceroute to www.ethz.ch (129.132.19.216), 30 hops max, 60 byte pac
 1  169.236.151.2 (169.236.151.2) [AS22323]  0.339 ms  0.330 ms  0.3
 2  10.7.1.177 (10.7.1.177) [*]  0.346 ms  0.307 ms  0.303 ms
 3  10.7.2.2 (10.7.2.2) [*]  0.861 ms  0.883 ms  0.859 ms
 4  10.7.2.18 (10.7.2.18) [*]  1.143 ms  1.067 ms  1.132 ms
 5  10.7.1.226 (10.7.1.226) [*]  11.272 ms  11.308 ms  11.338 ms
 6  hpr-tri-hpr3--ucm-10ge.cenic.net (137.164.27.113) [AS2152]  6.34
 7  hpr-riv-hpr3--tri-hpr3-100g.cenic.net (137.164.25.93) [AS2152]
 8  137.164.25.86 (137.164.25.86) [AS2152]  16.999 ms  17.022 ms  17
 9  hpr-lax-hpr3--sdg-hpr3-100ge.cenic.net (137.164.25.90) [AS2152]
10  hpr-i2--lax-hpr3-r-and-e.cenic.net (137.164.26.201) [AS2152]  17
11  ae-5.4079.rtsw.wash.net.internet2.edu (162.252.70.158) [AS11537]
12  internet2-gw.mx1.lon.uk.geant.net (62.40.124.44) [AS21320/AS2096
13  ae6.mx1.lon2.uk.geant.net (62.40.98.37) [AS21320/AS20965]  152.7
14  ae5.mx1.par.fr.geant.net (62.40.98.179) [AS21320/AS20965]  159.2
15  ae5.mx1.gen.ch.geant.net (62.40.98.182) [AS21320/AS20965]  166.4
16  swice1-100ge-0-3-0-1.switch.ch (62.40.124.22) [AS21320/AS20965]
17  swiCE4-100GE-0-0-0-0.switch.ch (130.59.36.6) [AS559]  166.912 ms
18  swiBE3-100GE-0-1-0-1.switch.ch (130.59.37.145) [AS559]  169.068
19  swiBF1-100GE-0-0-0-1.switch.ch (130.59.39.78) [AS559]  169.329 m
20  swiEZ3-100GE-0-1-0-0.switch.ch (130.59.37.6) [AS559]  170.685 ms
21  rou-gw-lee-tengig-to-switch.ethz.ch (192.33.92.1) [AS559]  170.6
22  rou-fw-rz-rz-gw.ethz.ch (192.33.92.169) [AS559]  170.666 ms  170
23  www.ethz.ch (129.132.19.216) [AS559]  170.325 ms  170.305 ms  17
```

Aside: These are difficult problems

- Costs may change
  - If it's raining the cost of biking is higher
  - If a link is congested, the cost of routing a packet along it is higher
- The network might not be known
  - My computer doesn't store a map of the internet
- We want to do these tasks really quickly
  - I have time to bike to Savemart, but not to think about whether I should bike to Savemart…
  - More seriously, **the Internet.**

This is a joke.

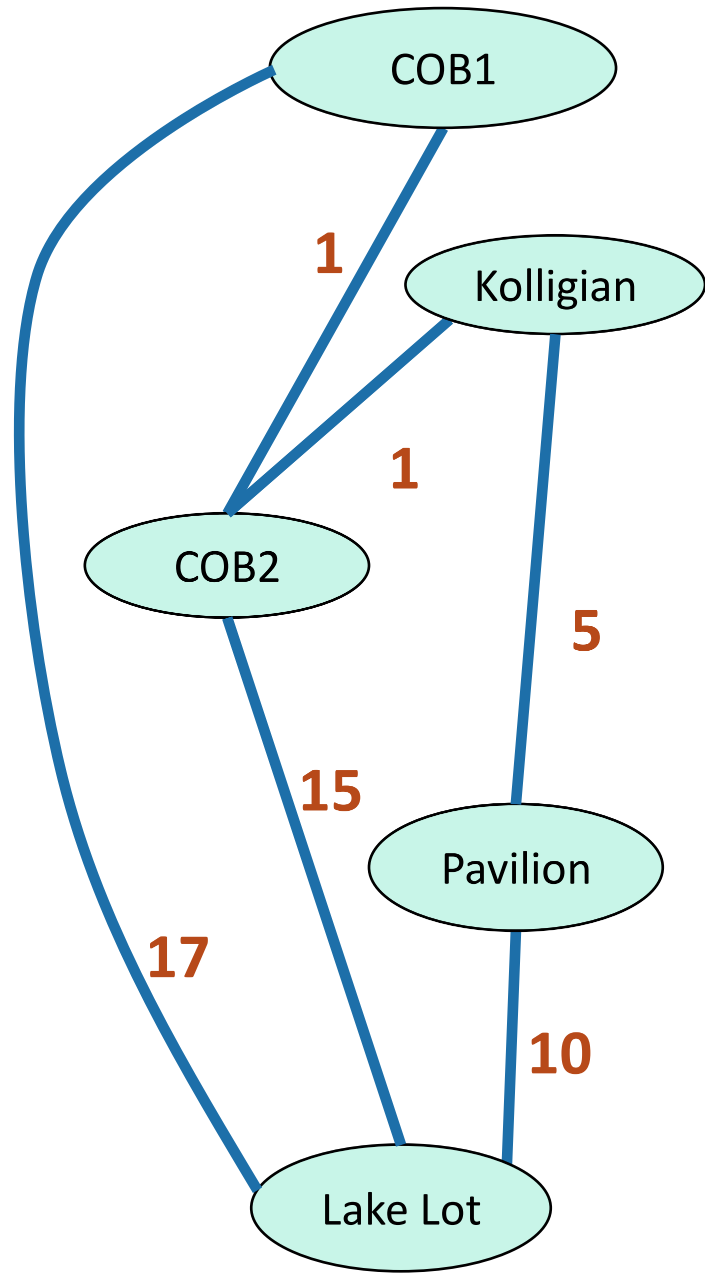
But let's ignore them for now.



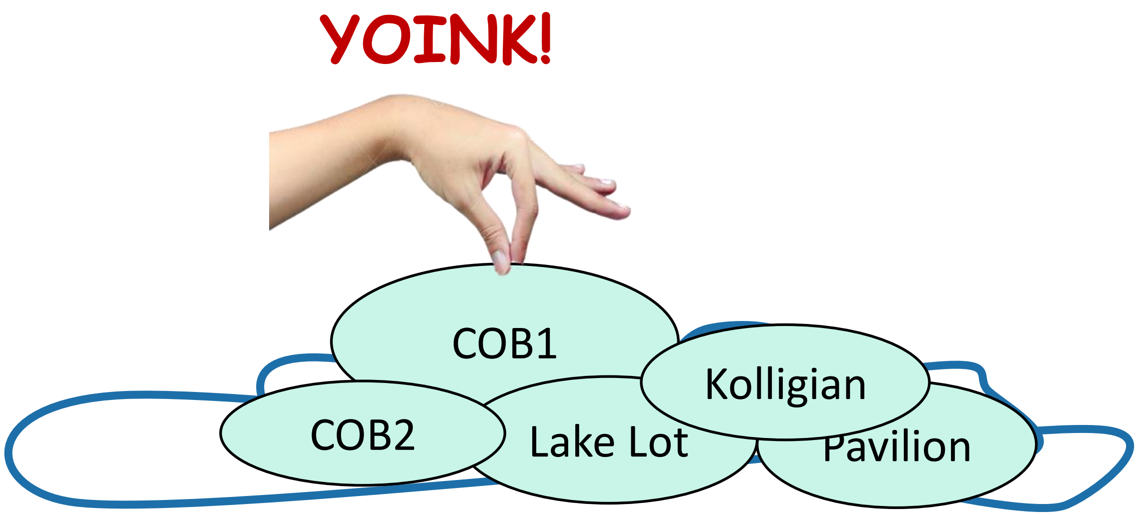
Back to this example

Dijkstra’s algorithm

- Finds shortest paths from COB1 to everywhere else.

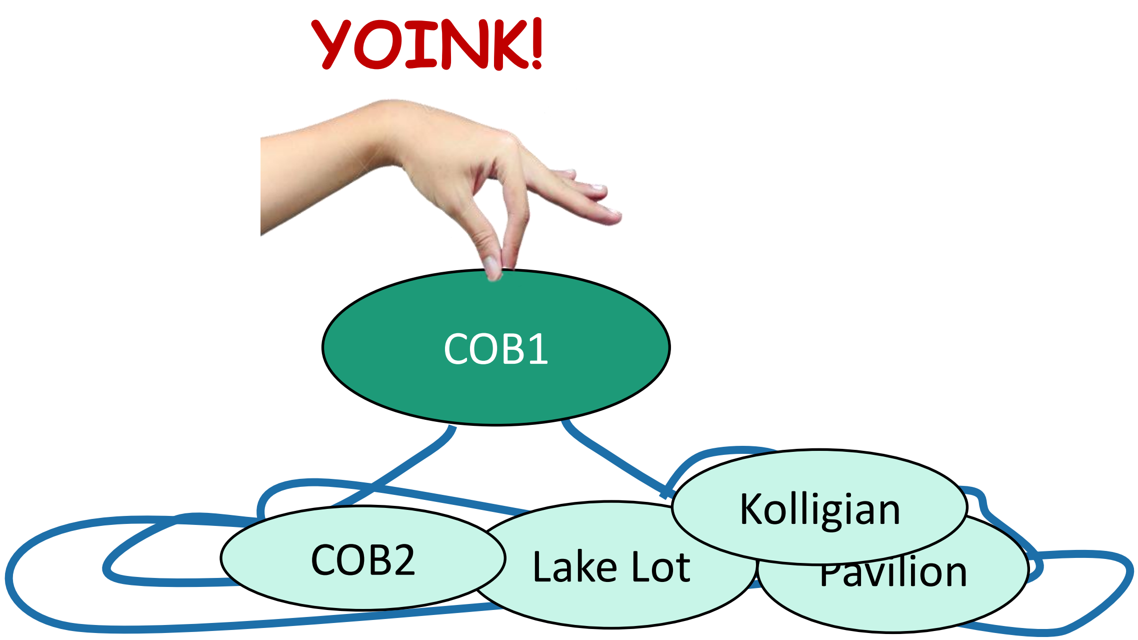


Dijkstra
intuition

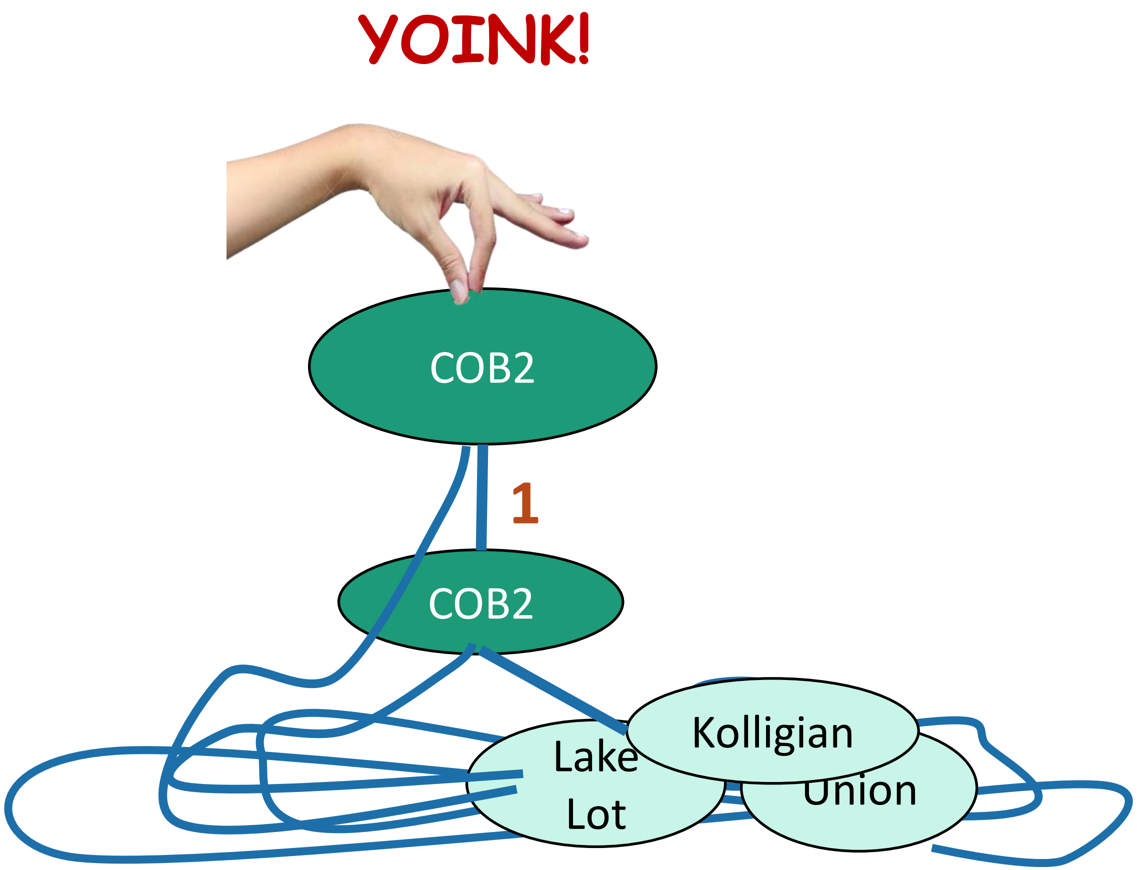


Dijkstra
intuition

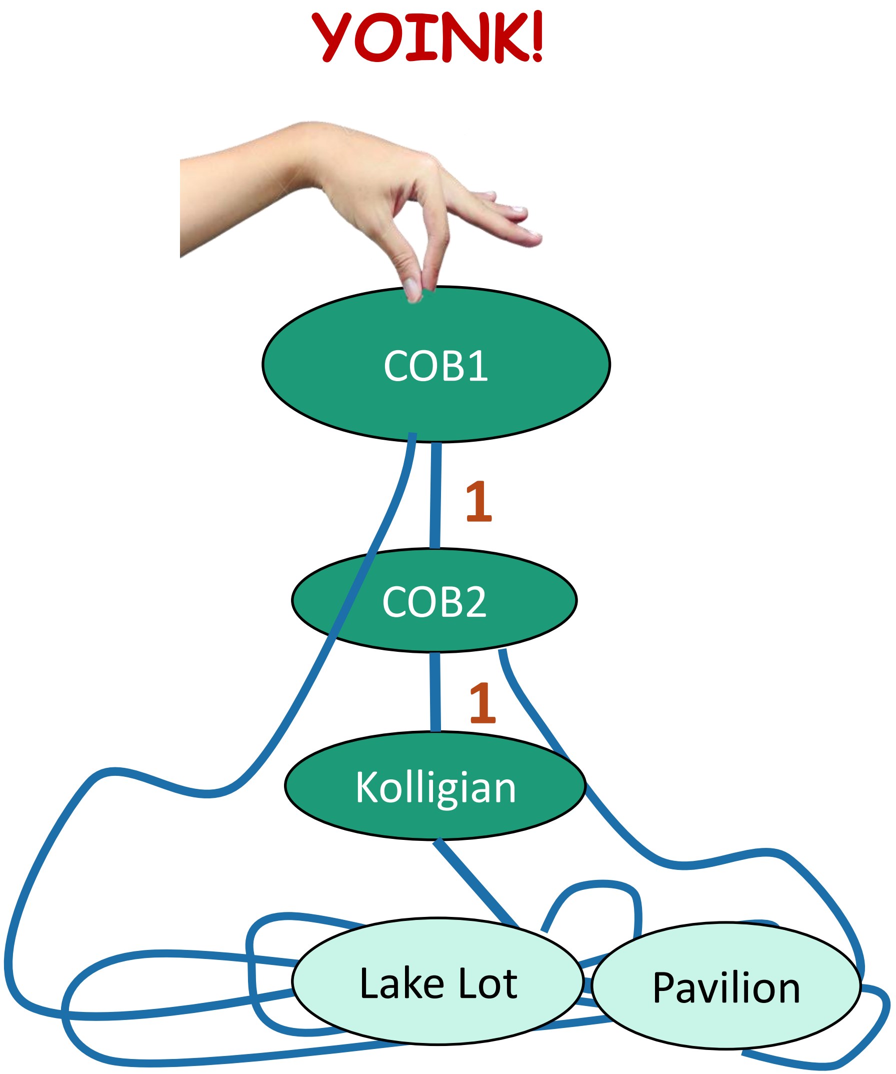
A vertex is done when it’s not on the ground anymore.



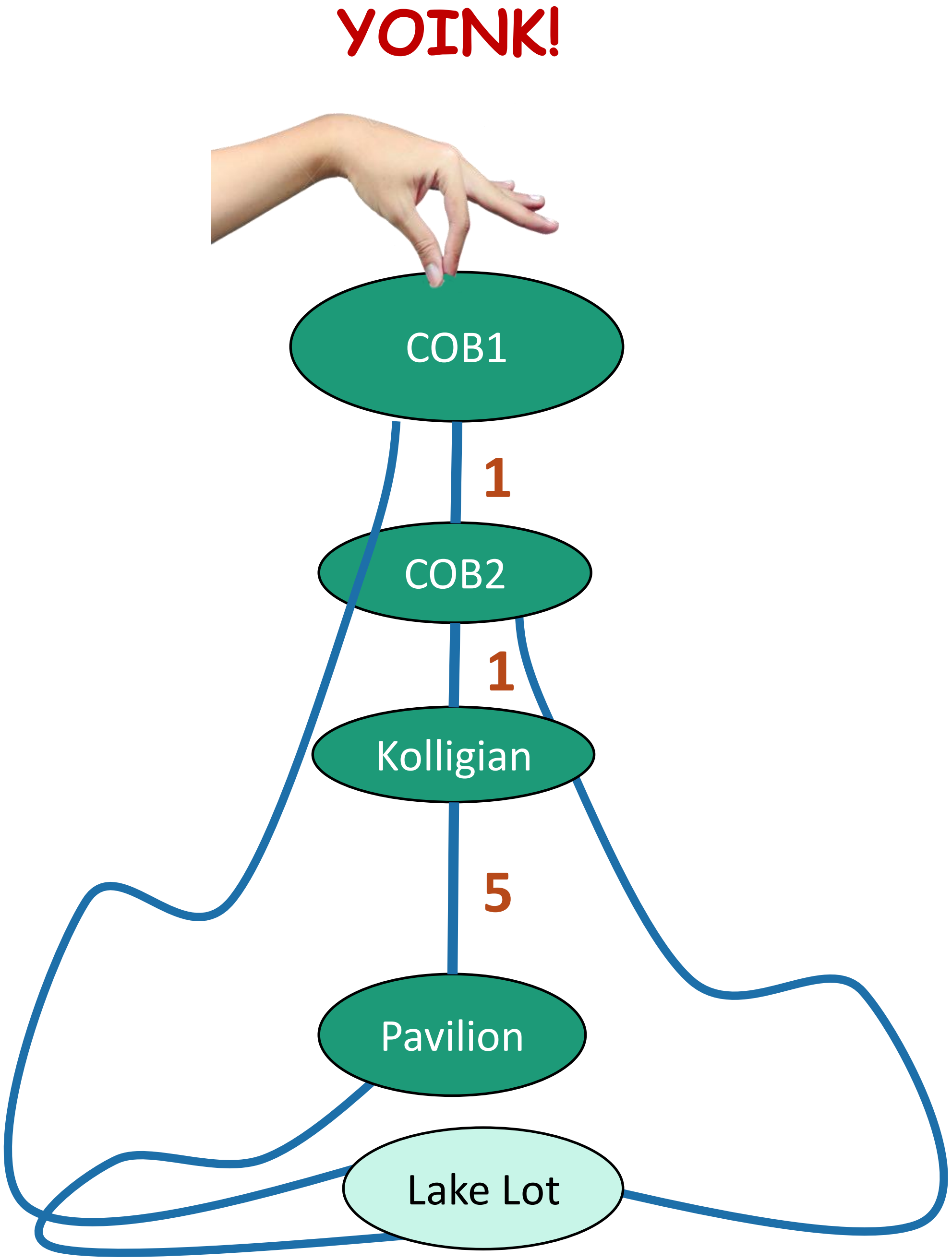
Dijkstra
intuition



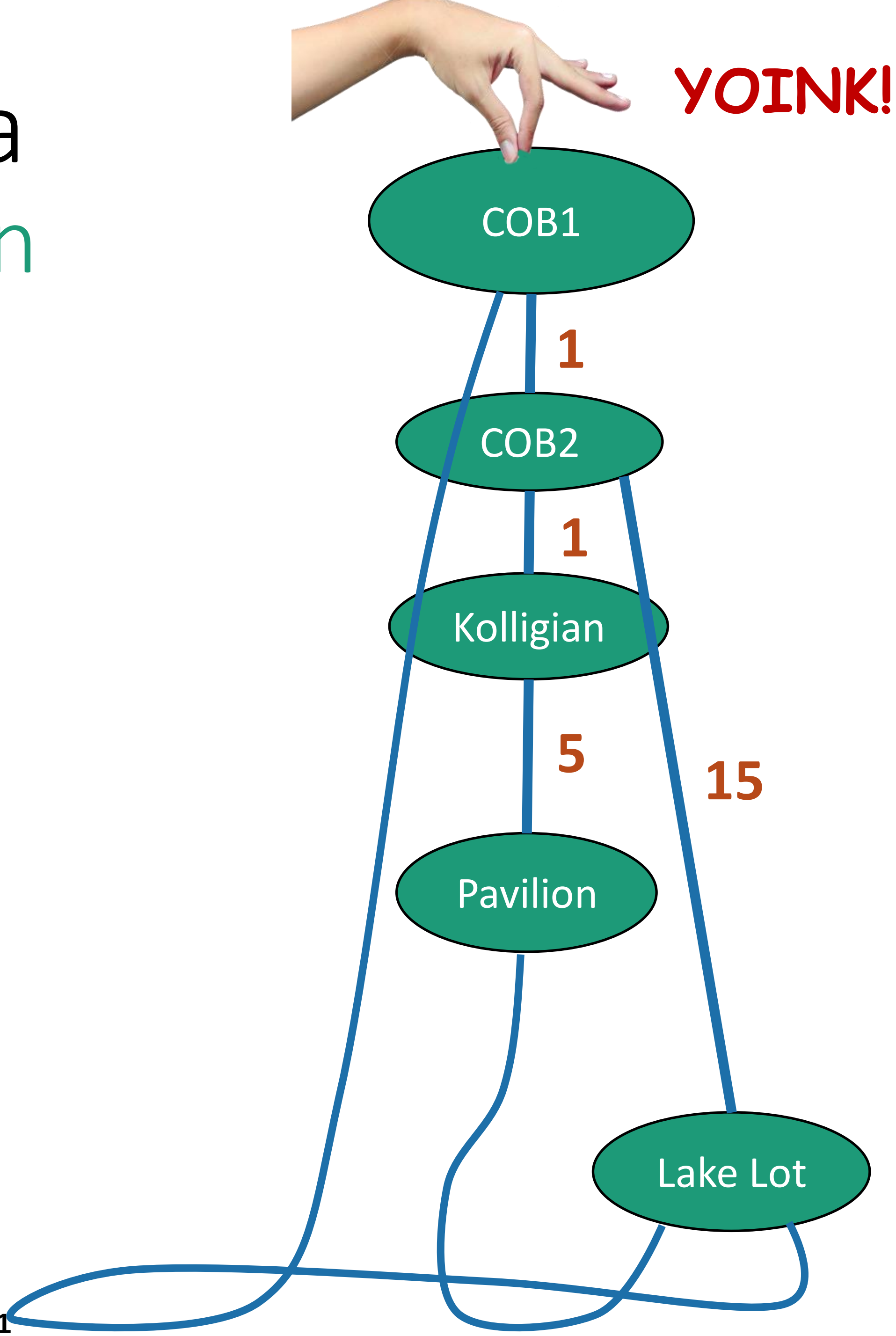
Dijkstra
intuition



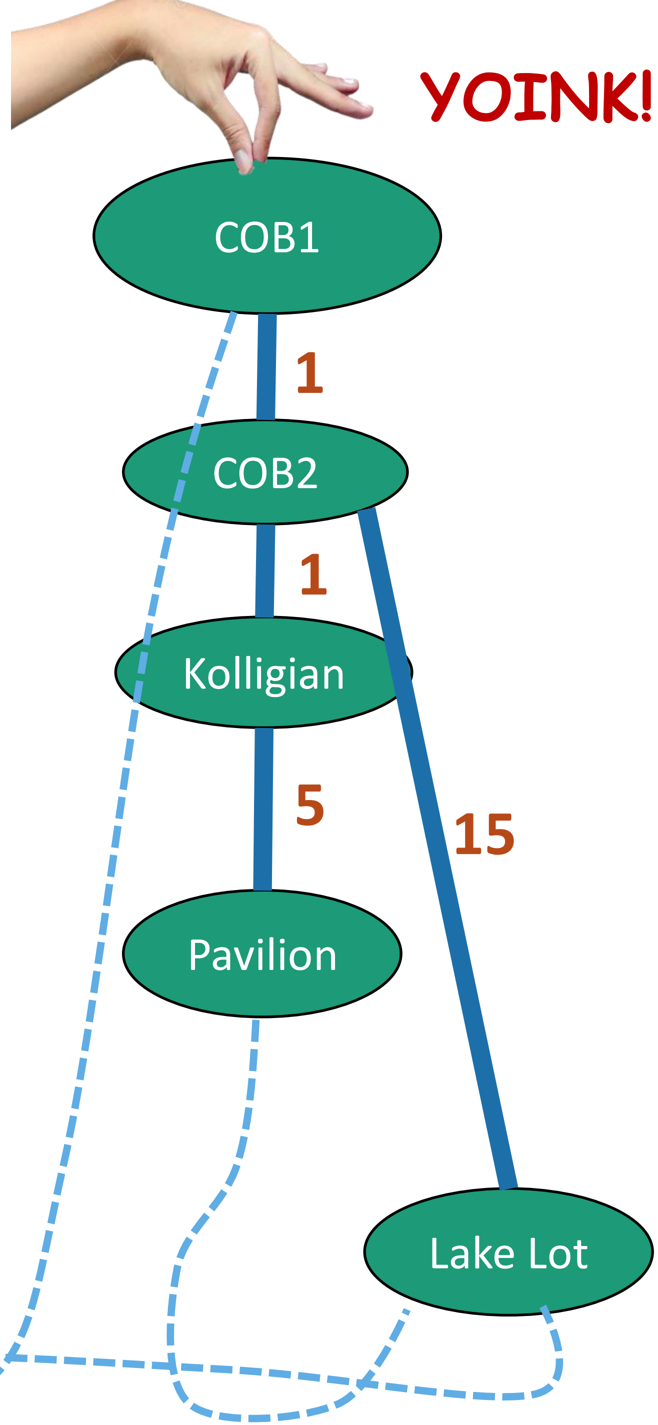
Dijkstra
intuition



Dijkstra
intuition



Dijkstra
intuition

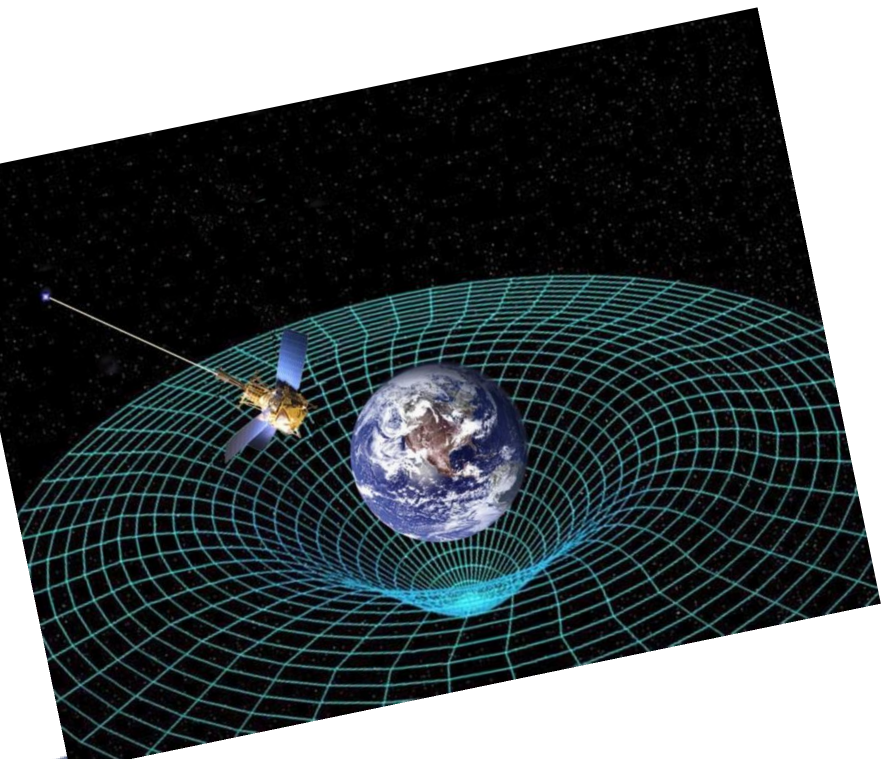


This creates a tree!

The shortest paths are the lengths along this tree.

How do we actually implement this?

- **Without** string and gravity?



Dijkstra by example

How far is a node from COB1?

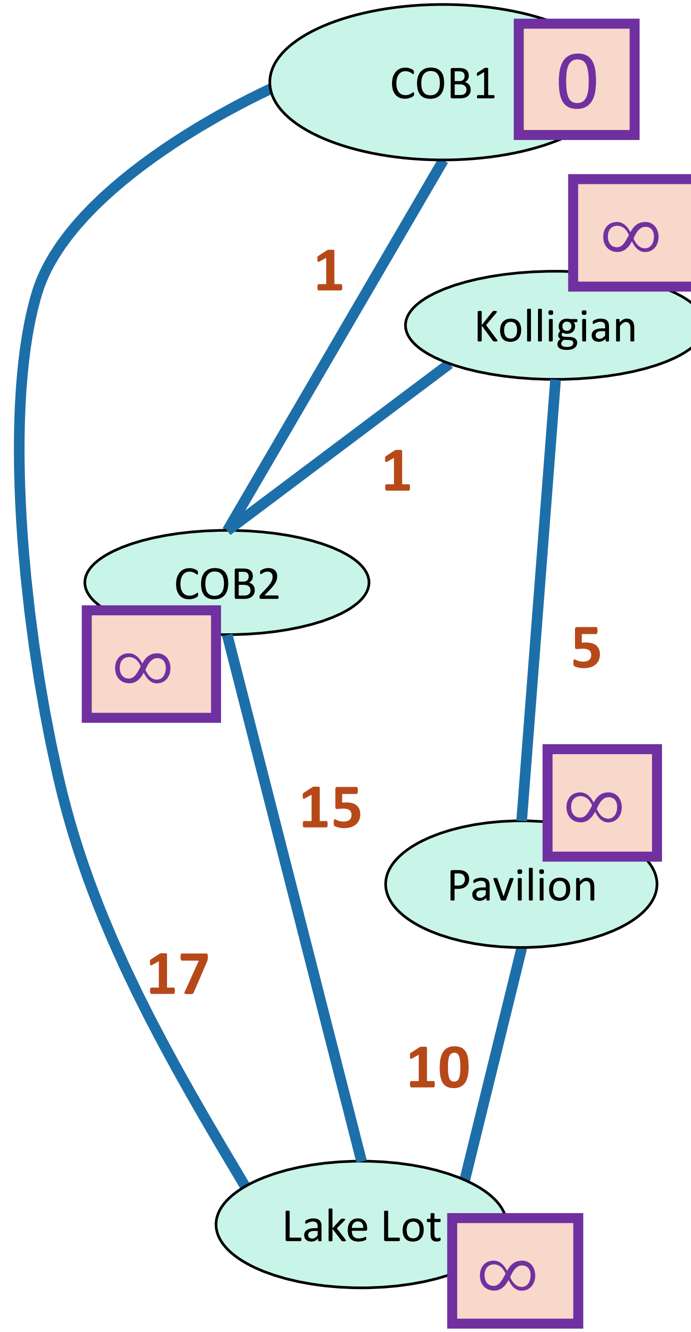
I'm not sure yet

I'm sure

x

x = d[v] is my best **over-estimate** for dist(COB1,v).

Initialize d[v] = ∞
for all non-starting vertices
v, and d[COB1] = 0



- Pick the **not-sure** node u with the smallest estimate **d[u].**

Dijkstra by example

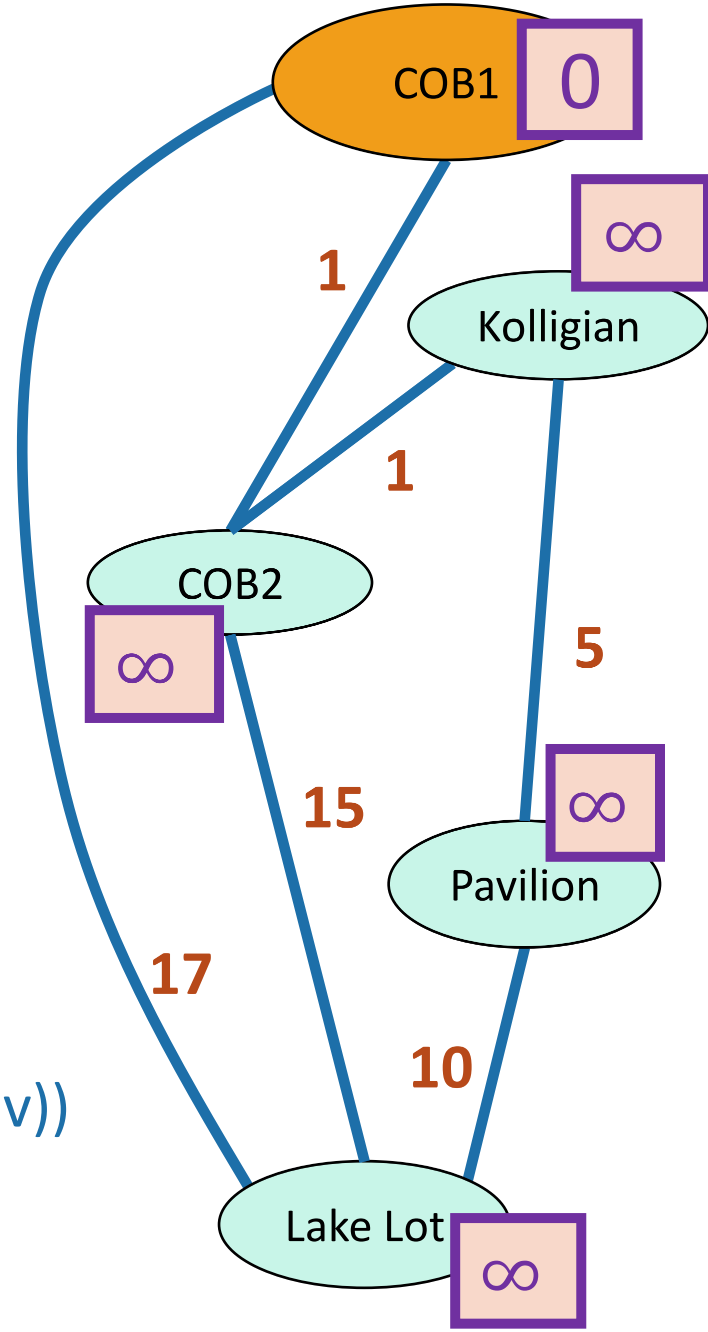
How far is a node from COB1?

I'm not sure yet

I'm sure

x — **x = d[v]** is my best **over-estimate** for dist(COB1,v).

Current node u



- Pick the **not-sure** node u with the smallest estimate **d[u].**
- Update all u's neighbors v:
  - d[v] = min(d[v] , d[u] + edgeWeight(u,v))

Dijkstra by example

How far is a node from COB1?

I'm not sure yet

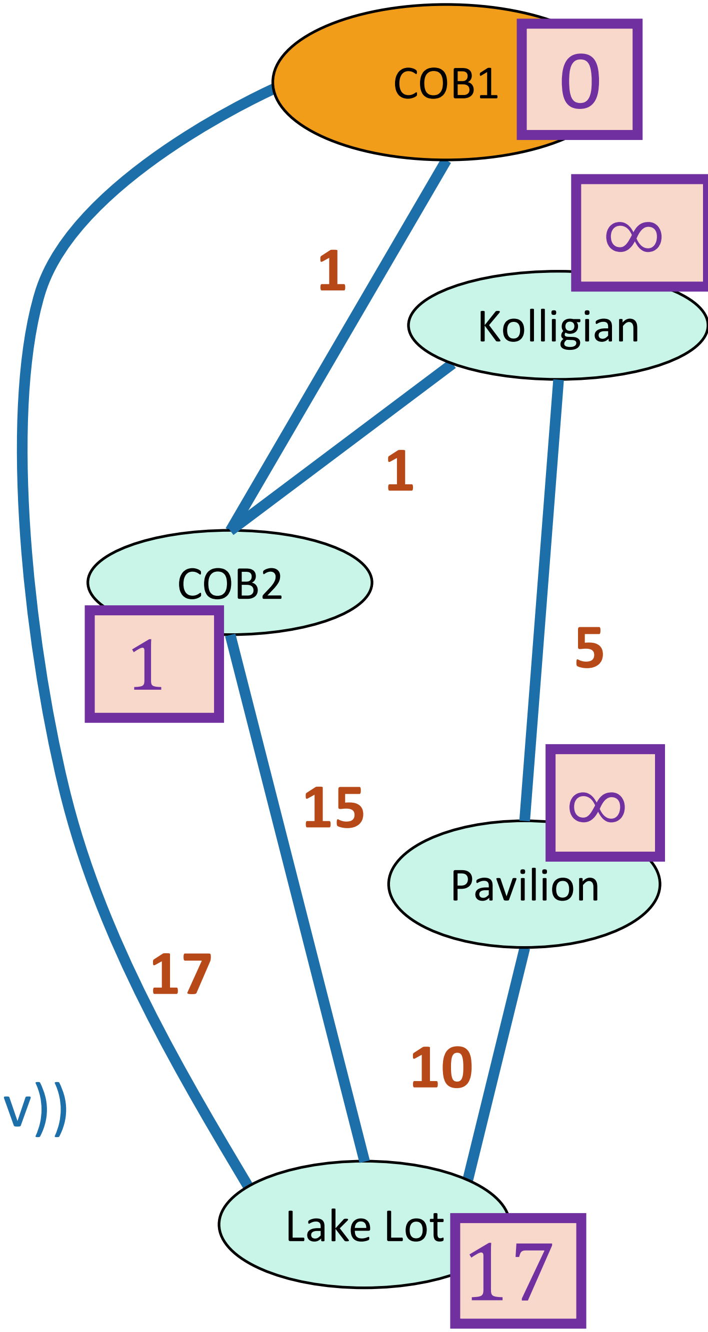
I'm sure

x

x = d[v] is my best **over-estimate** for dist(COB1,v).

Current node u

- Pick the **not-sure** node u with the smallest estimate **d[u].**
- Update all u's neighbors v:
  - d[v] = min(d[v] , d[u] + edgeWeight(u,v))
- Mark u as **sure**.



Dijkstra by example

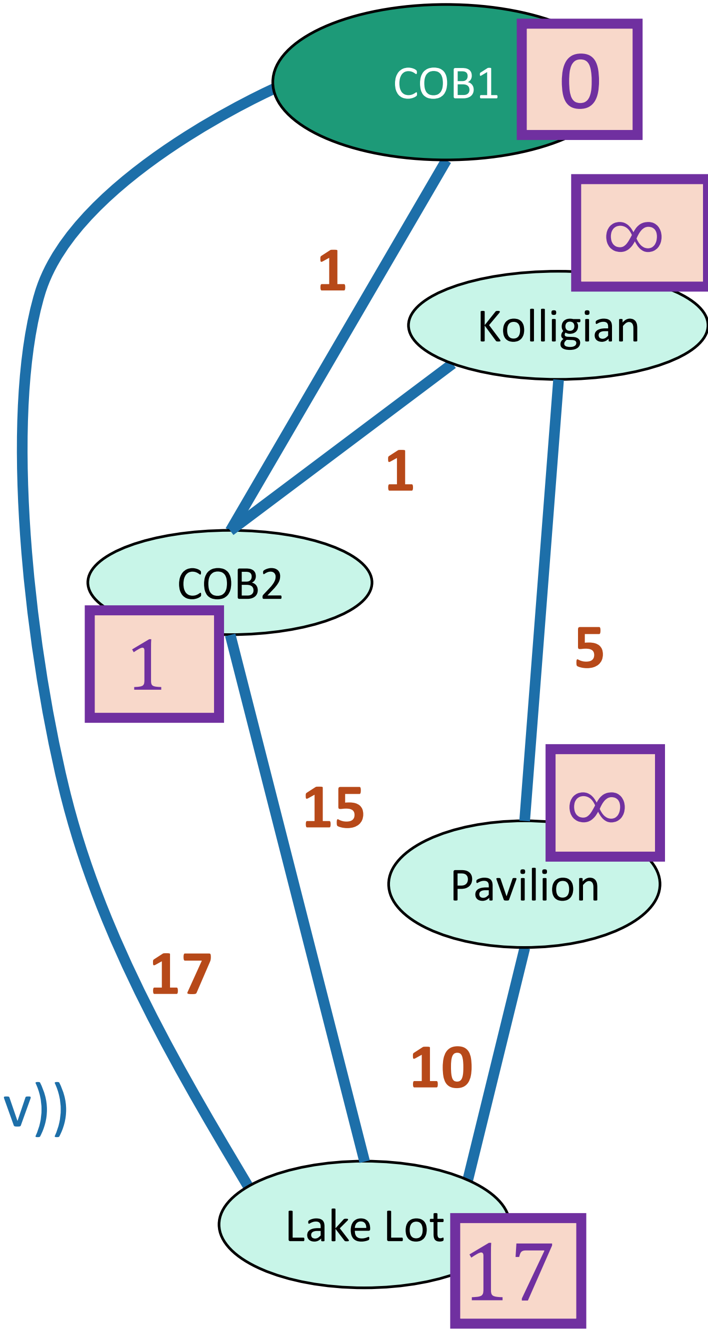
How far is a node from COB1?

I'm not sure yet

I'm sure

x **x = d[v]** is my best **over-estimate** for dist(COB1,v).

Current node u



- Pick the **not-sure** node u with the smallest estimate **d[u].**
- Update all u's neighbors v:
  - d[v] = min(d[v] , d[u] + edgeWeight(u,v))
- Mark u as **sure**.
- Repeat

Dijkstra by example

How far is a node from COB1?

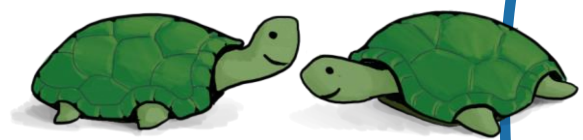
I'm not sure yet

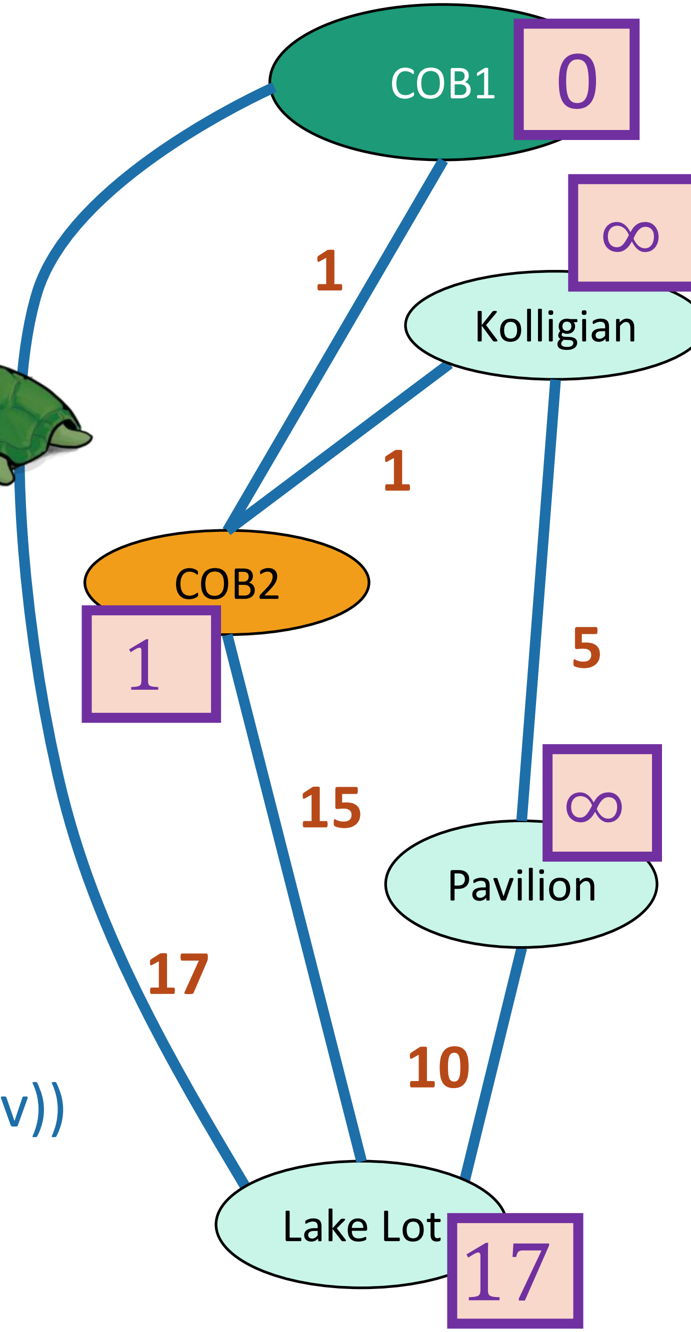
I'm sure

x — **x = d[v]** is my best **over-estimate** for dist(COB1,v).

Current node u

COB2 has three neighbors. What happens when we update them?





- Pick the **not-sure** node u with the smallest estimate **d[u].**
- Update all u's neighbors v:
  - d[v] = min(d[v] , d[u] + edgeWeight(u,v))
- Mark u as **sure**.
- Repeat

Dijkstra by example

How far is a node from COB1?

I'm not sure yet

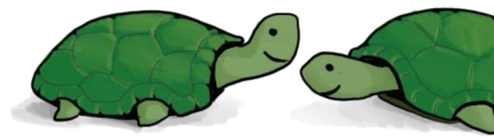
I'm sure

x

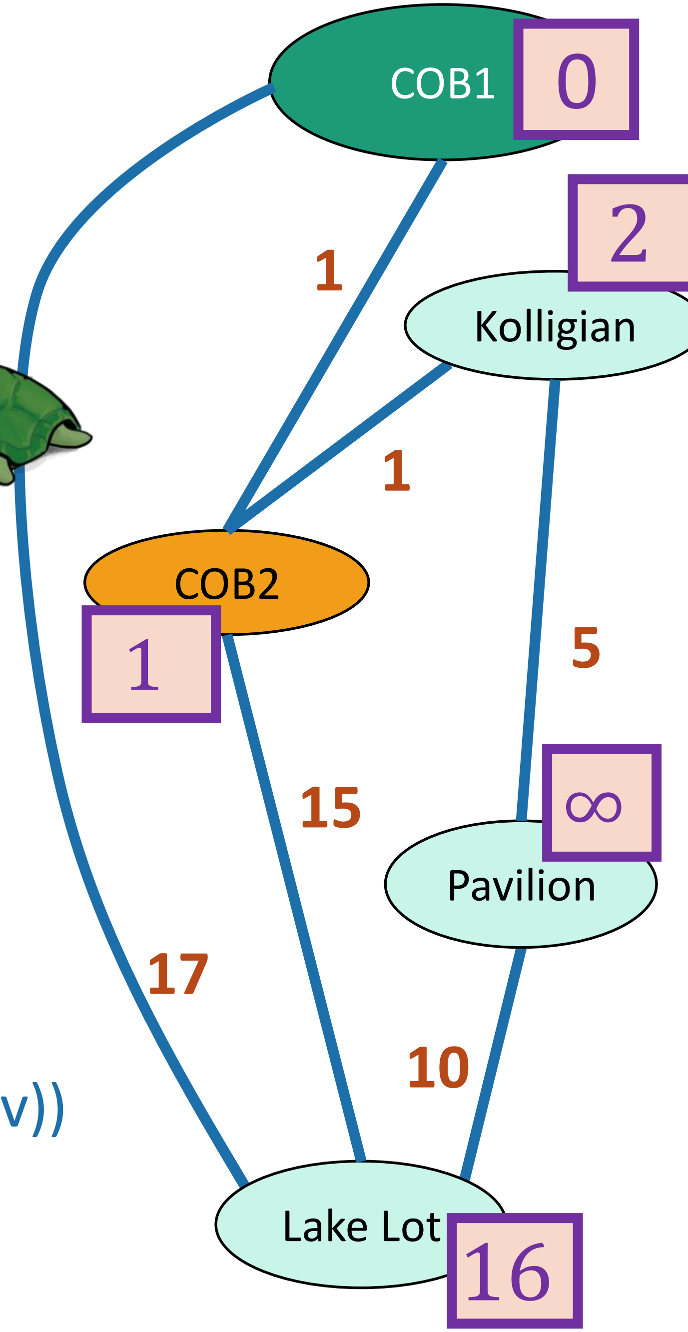
x = d[v] is my best **over-estimate** for dist(COB1,v).

Current node u

COB2 has three neighbors. What happens when we update them?

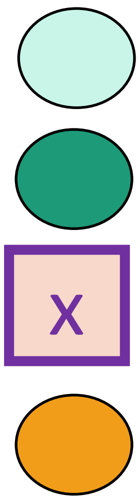


- Pick the **not-sure** node u with the smallest estimate **d[u].**
- Update all u's neighbors v:
  - d[v] = min(d[v] , d[u] + edgeWeight(u,v))
- Mark u as **sure**.
- Repeat



Dijkstra by example

How far is a node from COB1?



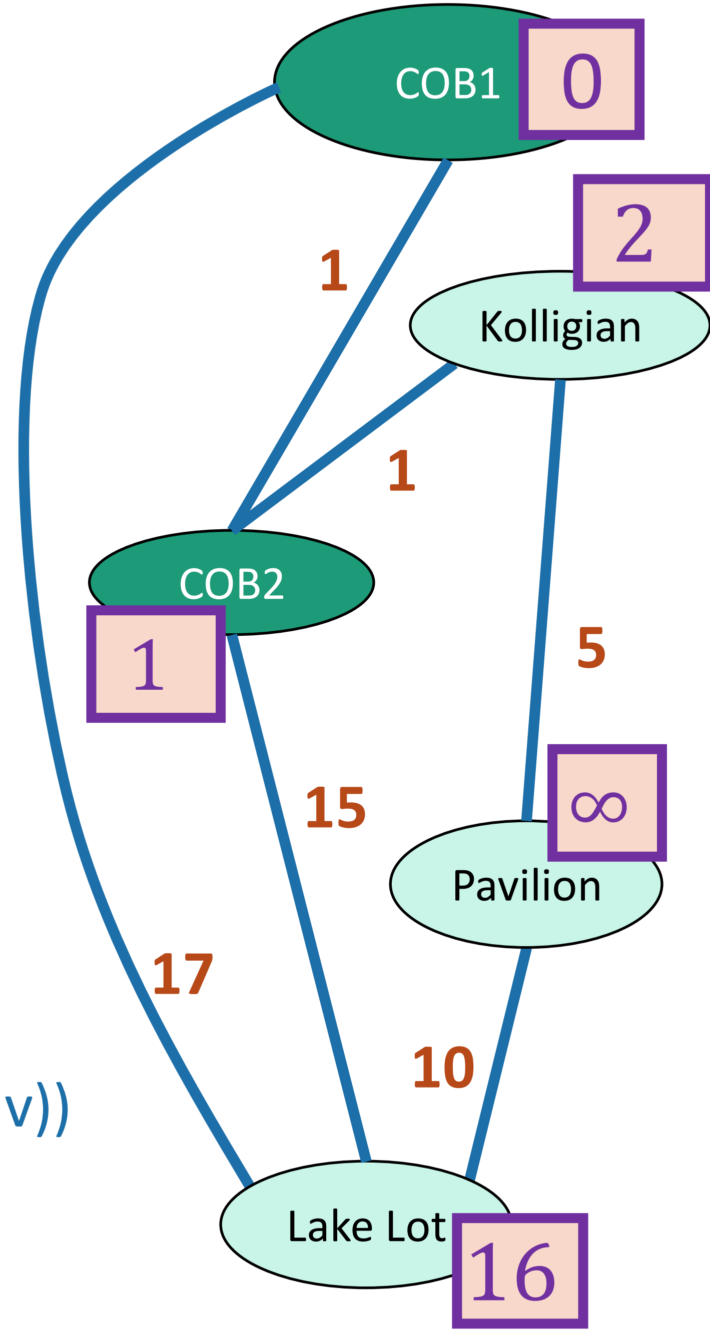
I'm not sure yet

I'm sure

x = d[v] is my best **over-estimate** for dist(COB1,v).

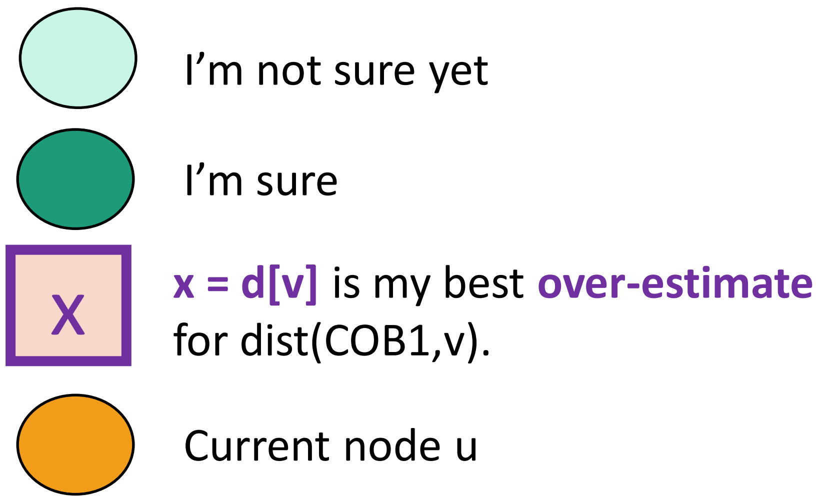
Current node u

- Pick the **not-sure** node u with the smallest estimate **d[u].**
- Update all u's neighbors v:
  - d[v] = min(d[v] , d[u] + edgeWeight(u,v))
- Mark u as **sure**.
- Repeat

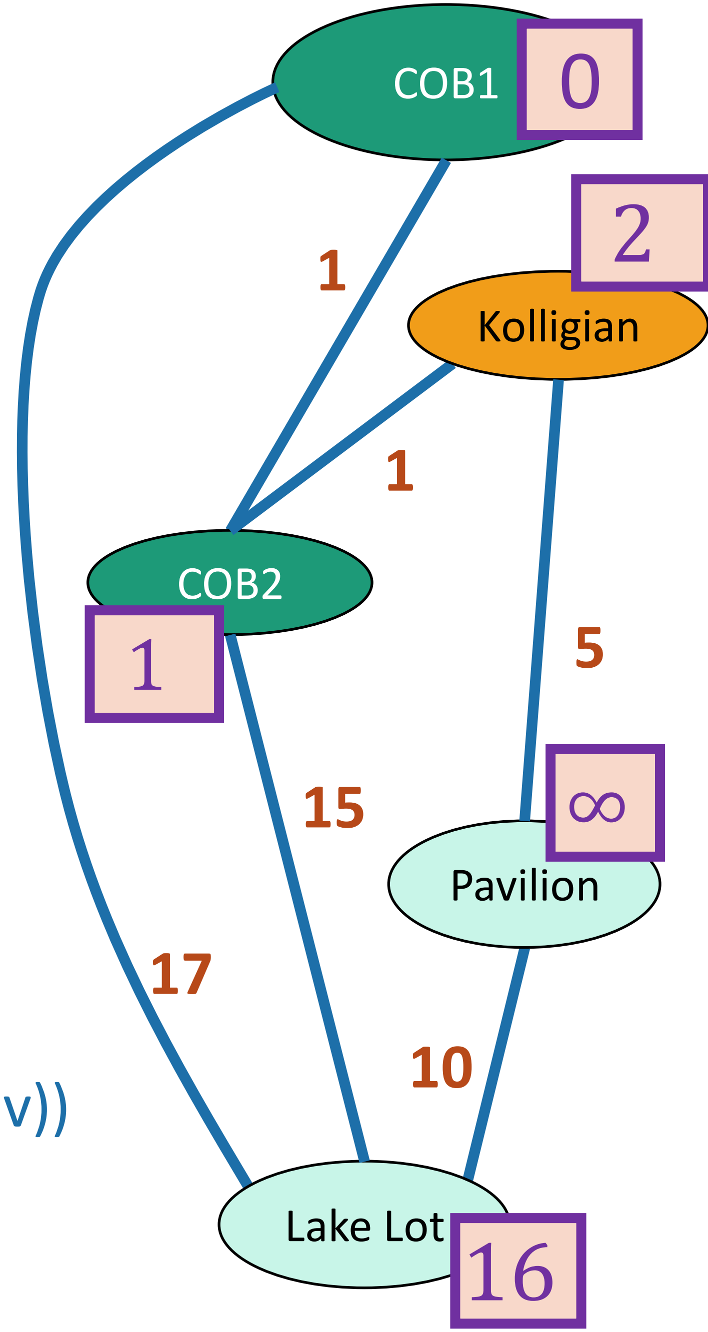


Dijkstra by example

How far is a node from COB1?



- Pick the **not-sure** node u with the smallest estimate **d[u].**
- Update all u's neighbors v:
  - d[v] = min(d[v] , d[u] + edgeWeight(u,v))
- Mark u as **sure**.
- Repeat



Dijkstra by example

How far is a node from COB1?

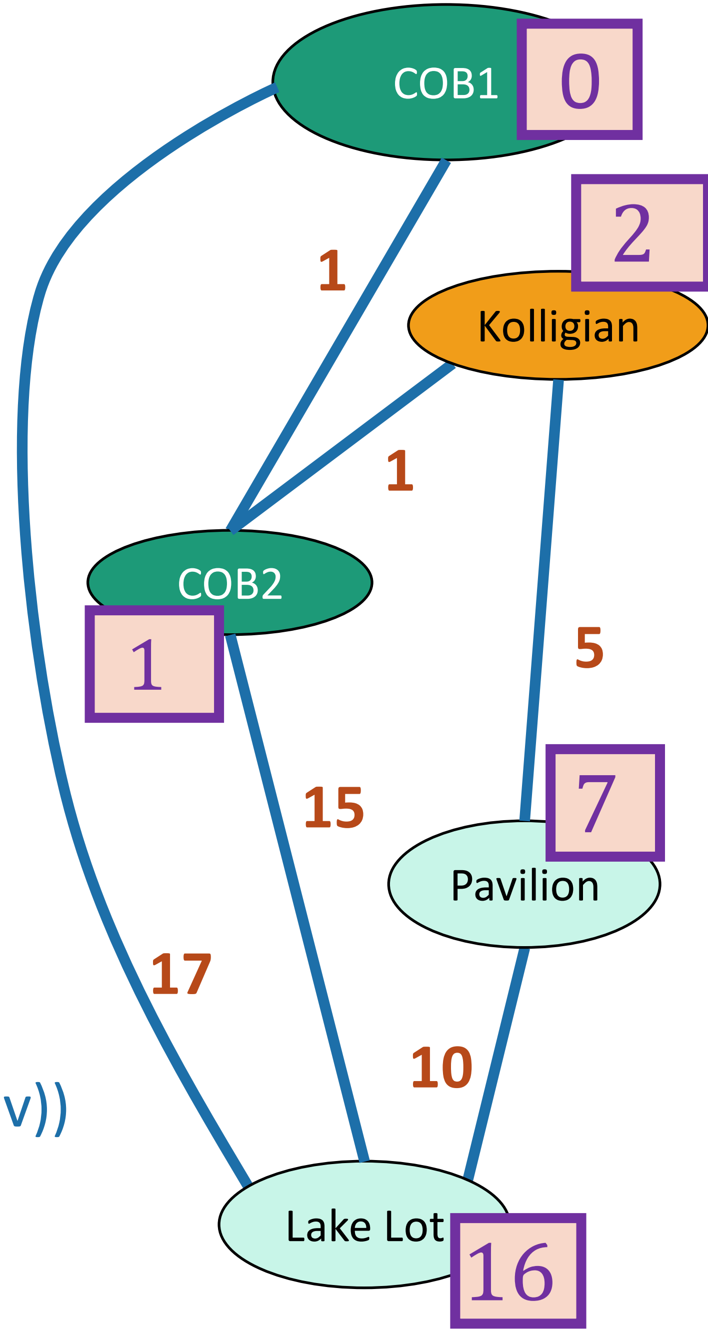
I'm not sure yet

I'm sure

x

x = d[v] is my best **over-estimate** for dist(COB1,v).

Current node u



- Pick the **not-sure** node u with the smallest estimate **d[u].**
- Update all u's neighbors v:
  - d[v] = min(d[v] , d[u] + edgeWeight(u,v))
- Mark u as **sure**.
- Repeat

Dijkstra by example

How far is a node from COB1?

I’m not sure yet

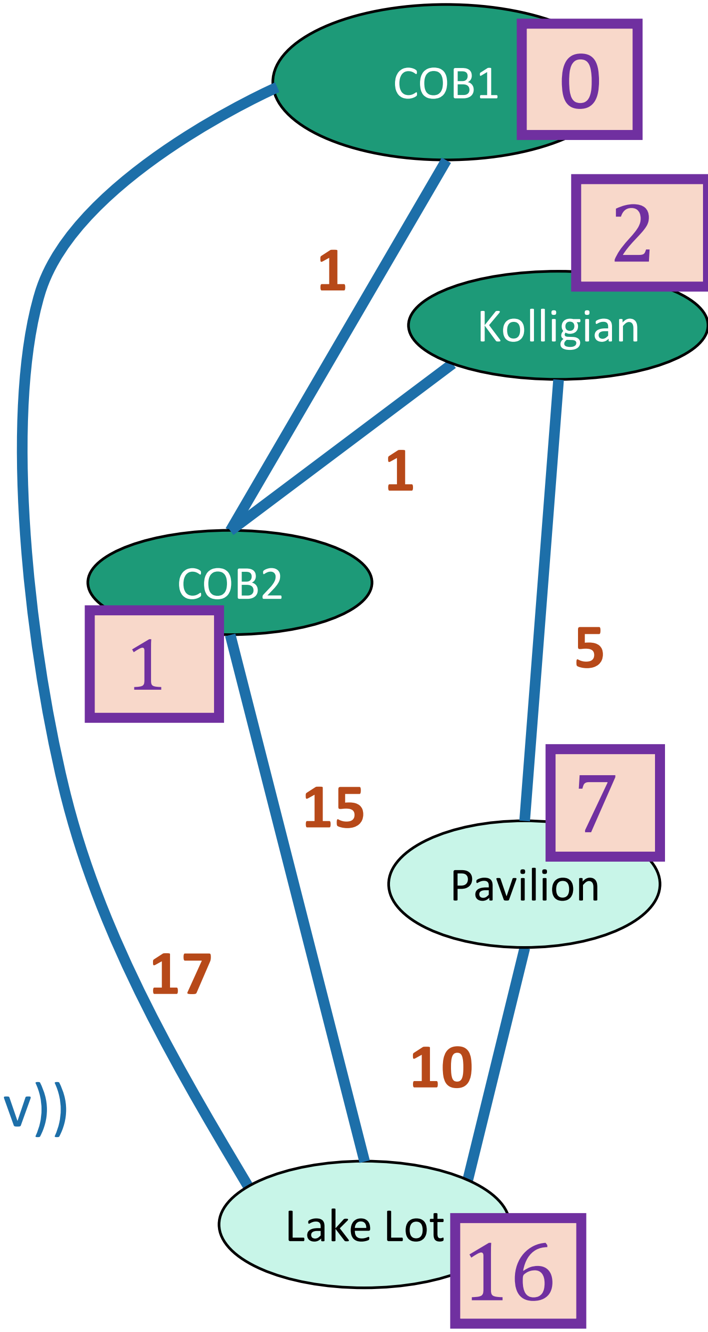
I’m sure

x

x = d[v] is my best **over-estimate** for dist(COB1,v).

Current node u

- Pick the **not-sure** node u with the smallest estimate **d[u].**
- Update all u’s neighbors v:
  - d[v] = min(d[v] , d[u] + edgeWeight(u,v))
- Mark u as **sure**.
- Repeat



Dijkstra by example

How far is a node from COB1?

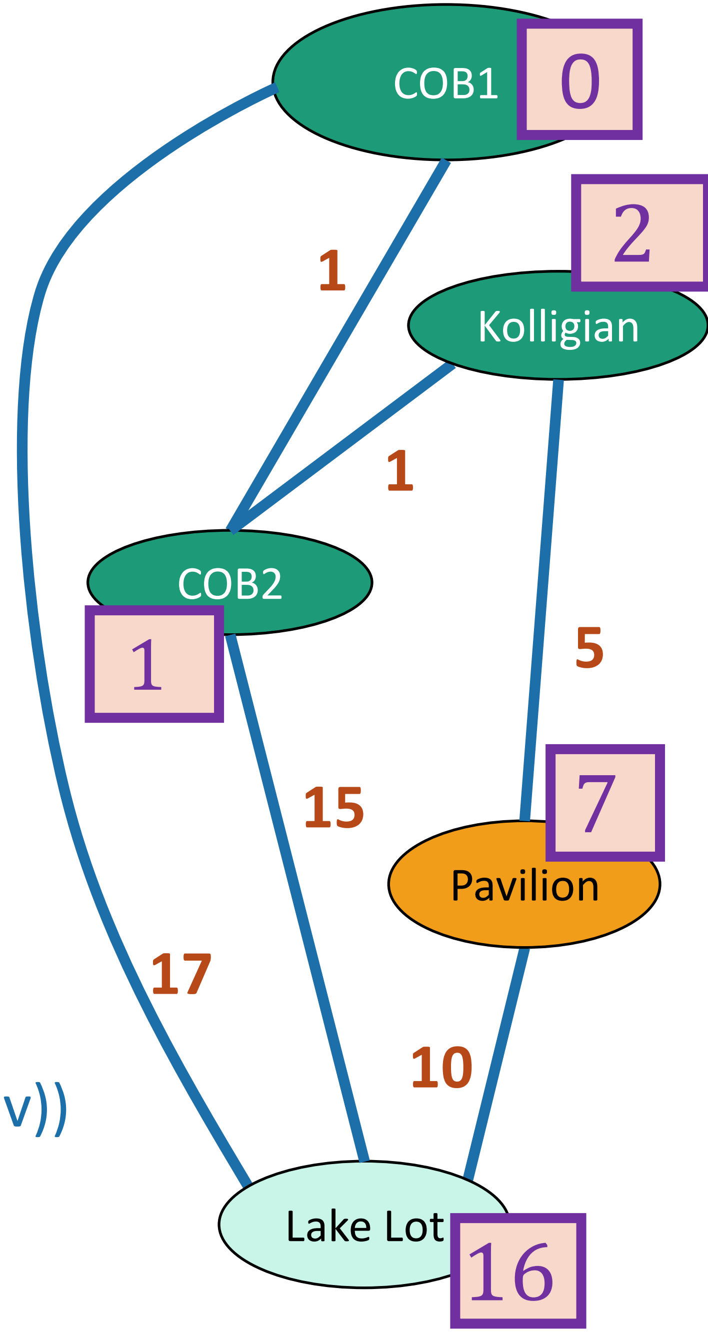
I'm not sure yet

I'm sure

x — **x = d[v]** is my best **over-estimate** for dist(COB1,v).

Current node u

- Pick the **not-sure** node u with the smallest estimate **d[u].**
- Update all u's neighbors v:
  - d[v] = min(d[v] , d[u] + edgeWeight(u,v))
- Mark u as **sure**.
- Repeat



Dijkstra by example

How far is a node from COB1?

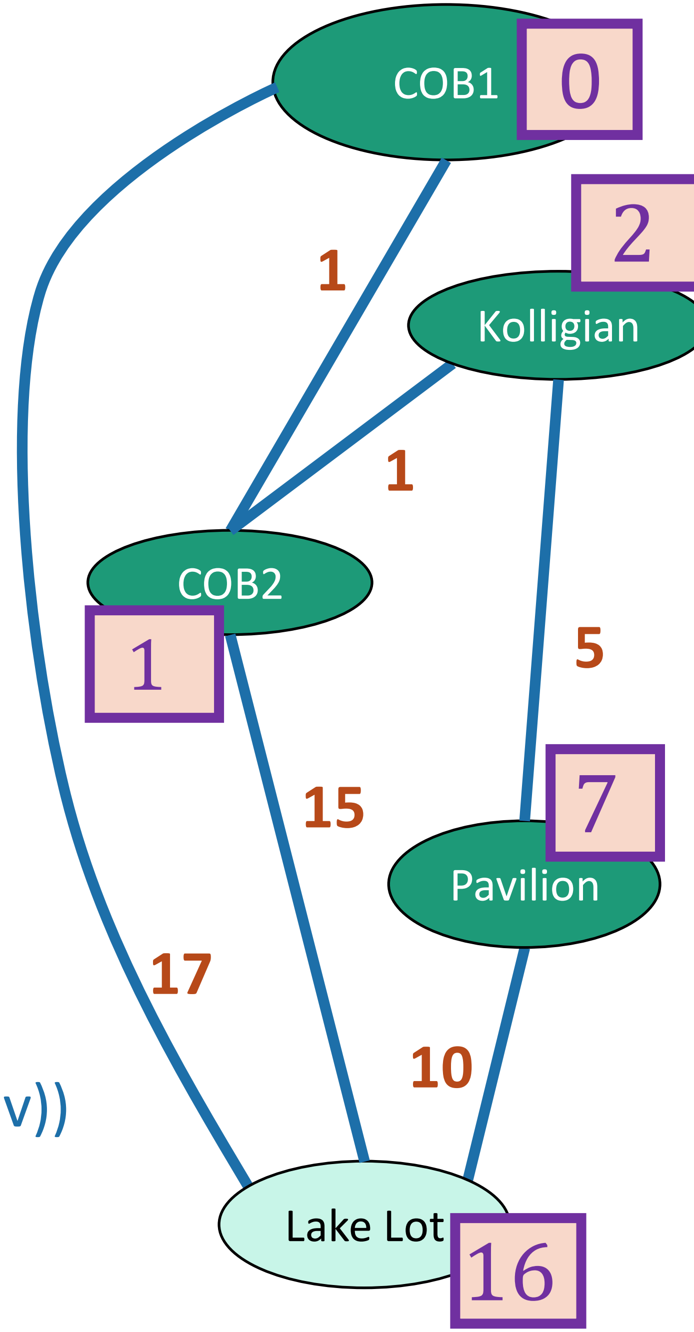
I'm not sure yet

I'm sure

x

x = d[v] is my best **over-estimate** for dist(COB1,v).

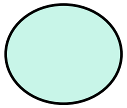
Current node u



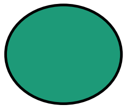
- Pick the **not-sure** node u with the smallest estimate **d[u].**
- Update all u's neighbors v:
  - d[v] = min(d[v] , d[u] + edgeWeight(u,v))
- Mark u as **sure**.
- Repeat

Dijkstra by example

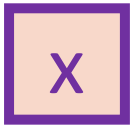
How far is a node from COB1?



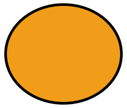
I'm not sure yet



I'm sure

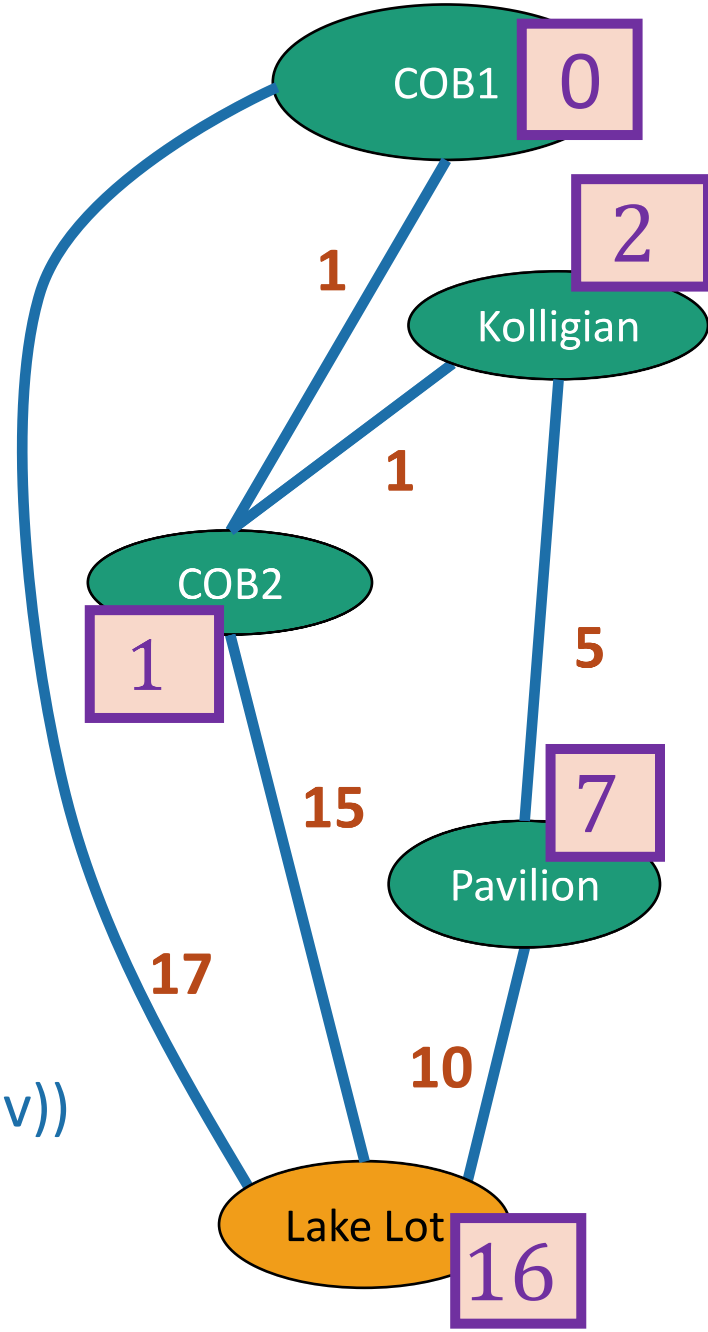


x = d[v] is my best **over-estimate** for dist(COB1,v).



Current node u

- Pick the **not-sure** node u with the smallest estimate **d[u].**
- Update all u's neighbors v:
  - d[v] = min(d[v] , d[u] + edgeWeight(u,v))
- Mark u as **sure**.
- Repeat



Dijkstra by example

How far is a node from COB1?

I'm not sure yet

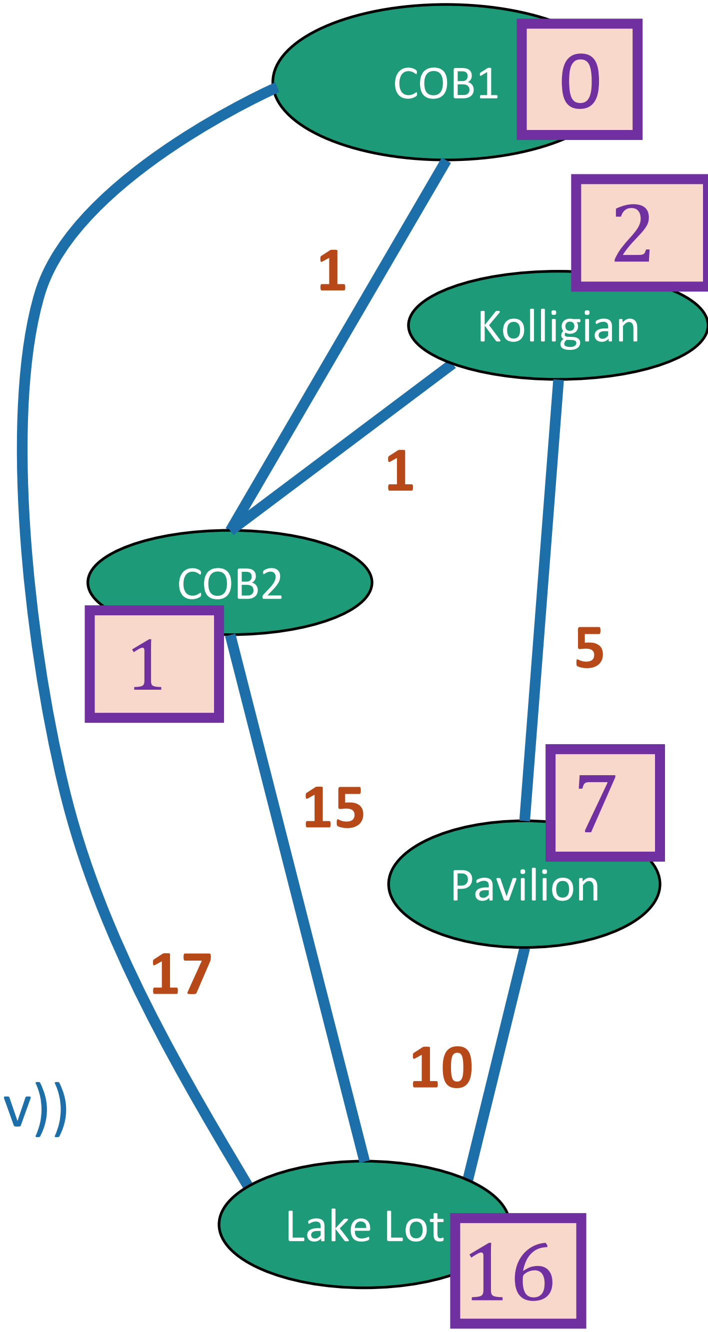
I'm sure

x

x = d[v] is my best **over-estimate** for dist(COB1,v).

Current node u

- Pick the **not-sure** node u with the smallest estimate **d[u].**
- Update all u's neighbors v:
  - d[v] = min(d[v] , d[u] + edgeWeight(u,v))
- Mark u as **sure**.
- Repeat



Dijkstra by example

How far is a node from COB1?

I'm not sure yet

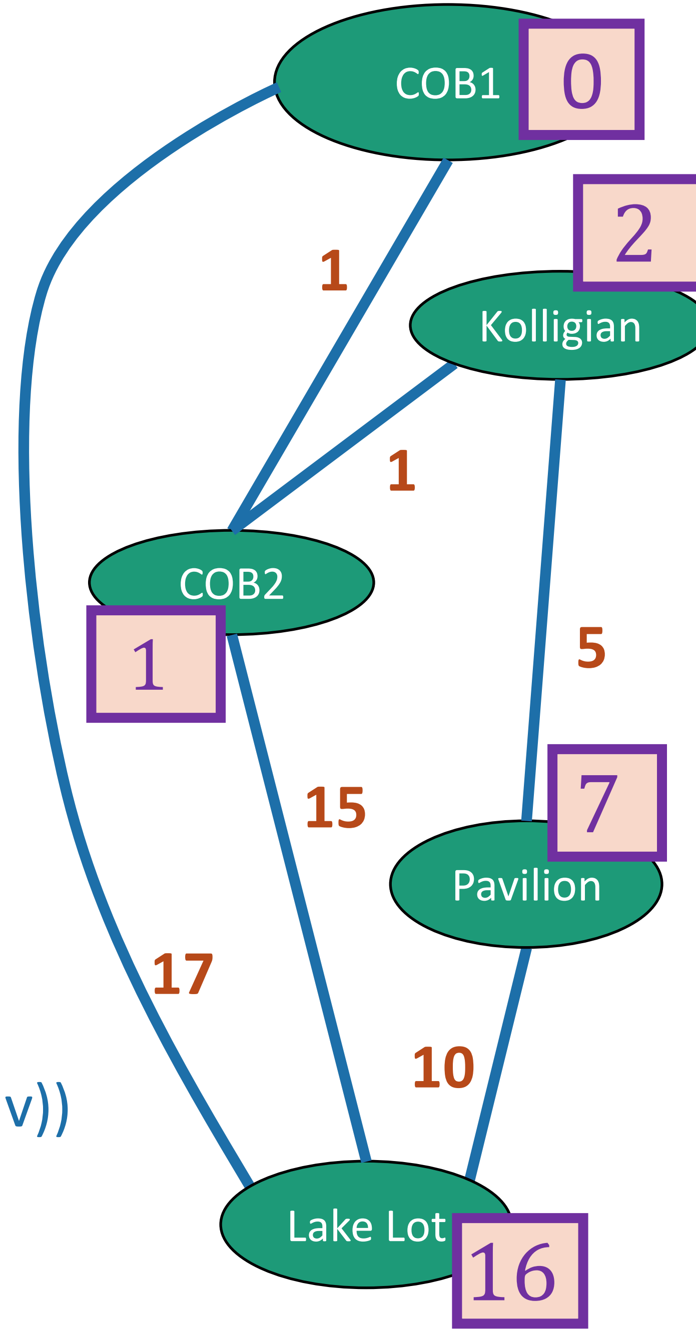
I'm sure

x

x = d[v] is my best **over-estimate** for dist(COB1,v).

Current node u

- Pick the **not-sure** node u with the smallest estimate **d[u].**
- Update all u's neighbors v:
  - d[v] = min(d[v] , d[u] + edgeWeight(u,v))
- Mark u as **sure**.
- Repeat



- After all nodes are **sure**, say that d(COB1, v) = d[v] for all v

Dijkstra’s algorithm

Dijkstra(G,s):

- Set all vertices to **not-sure**
- d[v] = ∞ for all v in V
- d[s] = 0
- **While** there are **not-sure** nodes:
  - Pick the **not-sure** node u with the smallest estimate **d[u].**
  - **For** v in u.neighbors:
    - d[v] ← min(d[v] , d[u] + edgeWeight(u,v))
  - Mark u as **sure**.
- Now d(s, v) = d[v]

Lots of implementation details left un-explained.
We’ll get to that!

As usual

- Does it work?
  - Yes.

- Is it fast?
  - Depends on how you implement it.

Why does this work?

- **Theorem**:
  - Suppose we run Dijkstra on G =(V,E), starting from s.
  - At the end of the algorithm, the estimate **d[v]** is the actual distance d(s,v).

Let's rename "COB1" to "s", our starting vertex.

- Proof outline:
  - **Claim 1**: For all v, **d[v] ≥ d(s,v).**
  - **Claim 2**: When a vertex v is marked **sure**, **d[v] = d(s,v).**

- **Claims 1 and 2** imply the **theorem.**
  - When v is marked **sure**, **d[v] = d(s,v).** Claim 2
  - **d[v] ≥ d(s,v)** and never increases, so after v is **sure**, **d[v]** stops changing. Claim 1 + def of algorithm
  - This implies that at any time *after* v is marked **sure**, **d[v] = d(s,v).**
  - All vertices are **sure** at the end, so all vertices end up with **d[v] = d(s,v).**

Next let's prove the claims!

Claim 1

$d[v] \geq d(s,v)$ for all v.

Intuition!

Informally:

- Every time we update d[v], we have a path in mind:

$$d[v] \leftarrow \min(\, d[v]\, ,\, d[u] + \mathrm{edgeWeight}(u,v)\,)$$

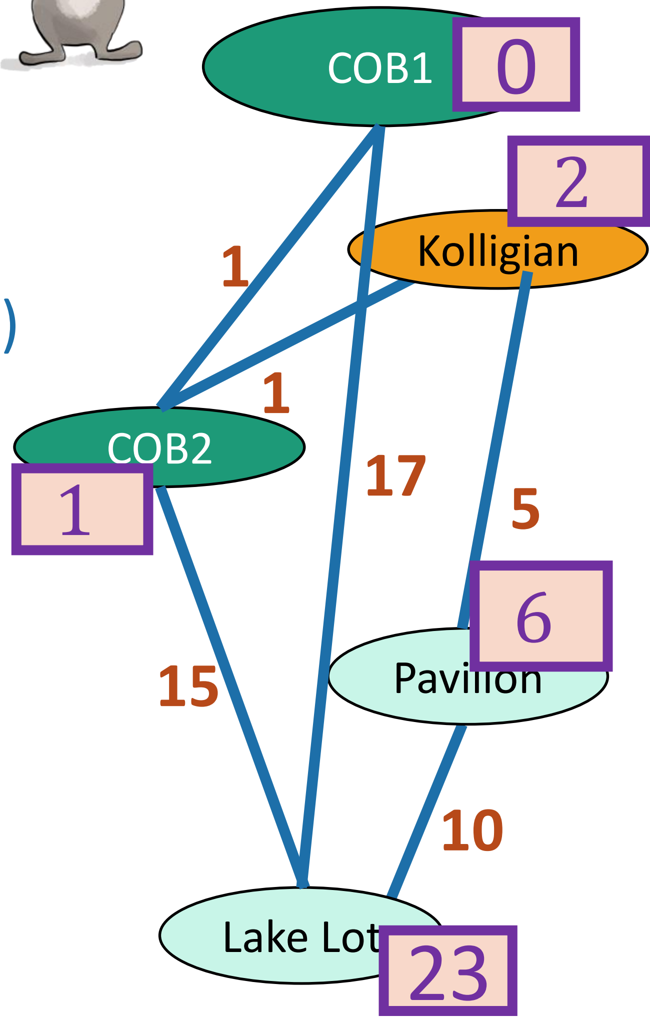
Whatever path we had in mind before

The shortest path to u, and then the edge from u to v.

- d[v] = length of the path we have in mind

  $\geq$ length of shortest path

  = d(s,v)



Formally:

- We should prove this by induction.
  - (See next slide or do it yourself)

Claim 1

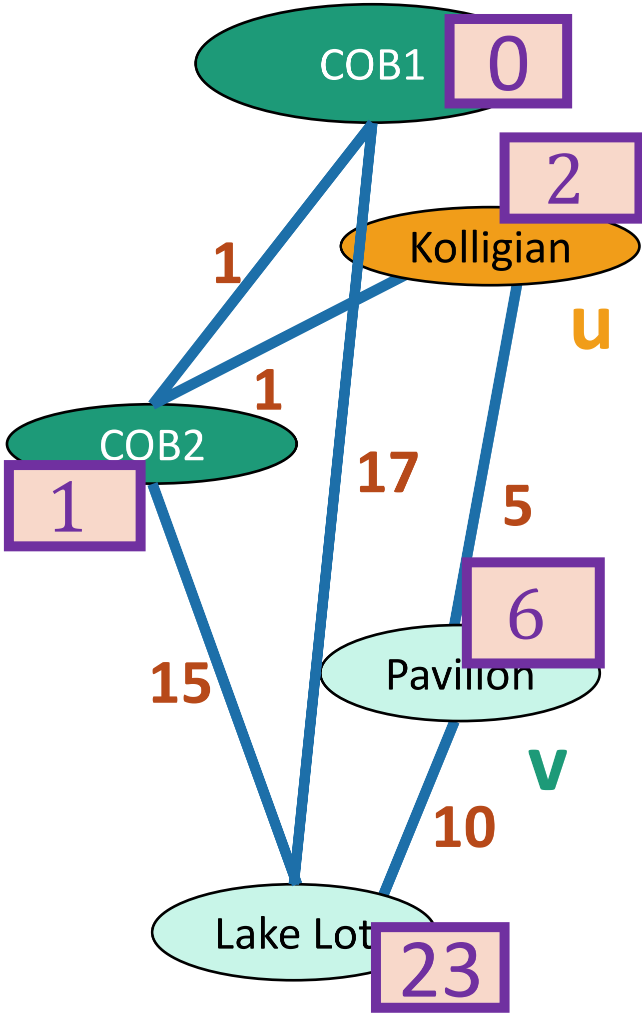
d[v] ≥ d(s,v) for all v.

- Inductive hypothesis.
  - After t iterations of Dijkstra, d[v] ≥ d(s,v) for all v.
- Base case:
  - At step 0, $\mathrm{d}(s,s) = 0$, and $d(s,v) \leq \infty$
- Inductive step: say hypothesis holds for t.
  - At step t+1:
    - Pick **u**; for each neighbor **v**:
    - d[v] ← min(d[v] , d[u] + w(u,v))

By induction,
$d(s,v) \leq d[v]$

$$\mathrm{d}(s,v) \leq d(s,u) + d(u,v)$$
$$\leq d[u] + w(u,v)$$

using induction again for d[u]



So the inductive hypothesis holds for t+1, and Claim 1 follows.

Claim 2

When a vertex u is marked sure, d[u] = d(s,u)

- Inductive Hypothesis:
  - When we mark the t'th vertex v as **sure**, d[v] = d(s,v).
- Base case:
  - The first vertex marked **sure** is s, and d[s] = d(s,s) = 0.
- Inductive step:
  - Suppose that we are about to add u to the **sure** list.
  - That is, we picked u in the first line here:

(Note: we are assuming here that the edge weights are non-negative, so there's no way to sneakily get from s to s with cost less than zero!)

> - Pick the **not-sure** node u with the smallest estimate **d[u].**
> - Update all u's neighbors v:
>   - d[v] ← min(d[v] , d[u] + edgeWeight(u,v))
> - Mark u as **sure**.
> - Repeat

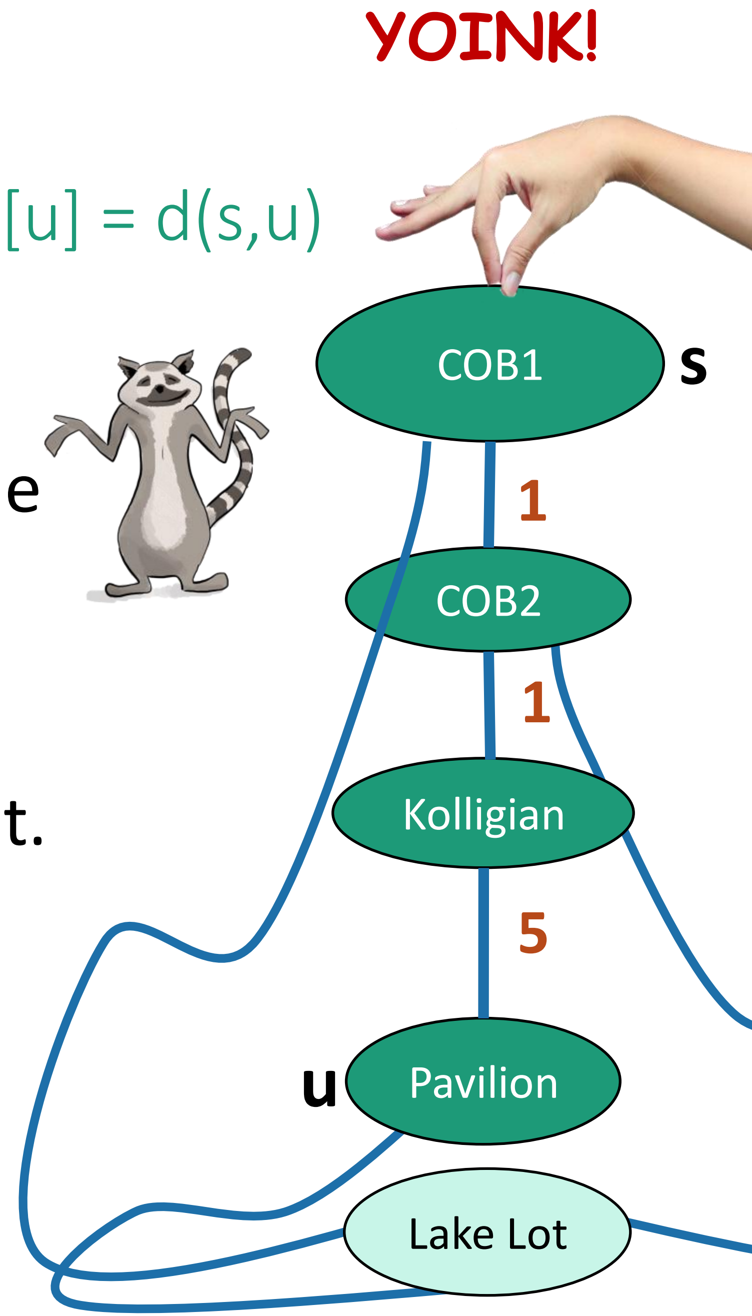
  - Assume by induction that every v already marked **sure** has d[v] = d(s,v).
  - Want to show that d[u] = d(s,u).

Intuition

When a vertex u is marked sure, d[u] = d(s,u)

- The first path that lifts **u** off the ground is the shortest one.

- But we should actually prove it.

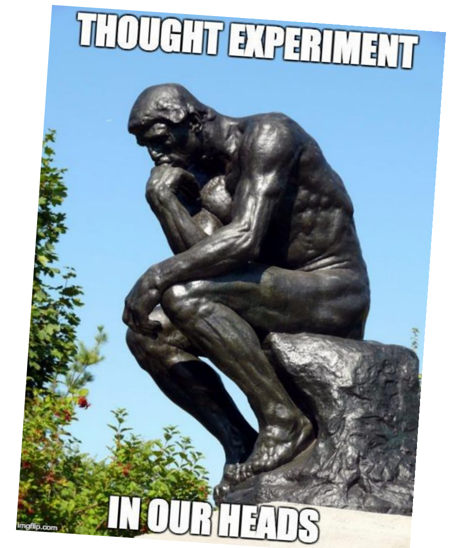


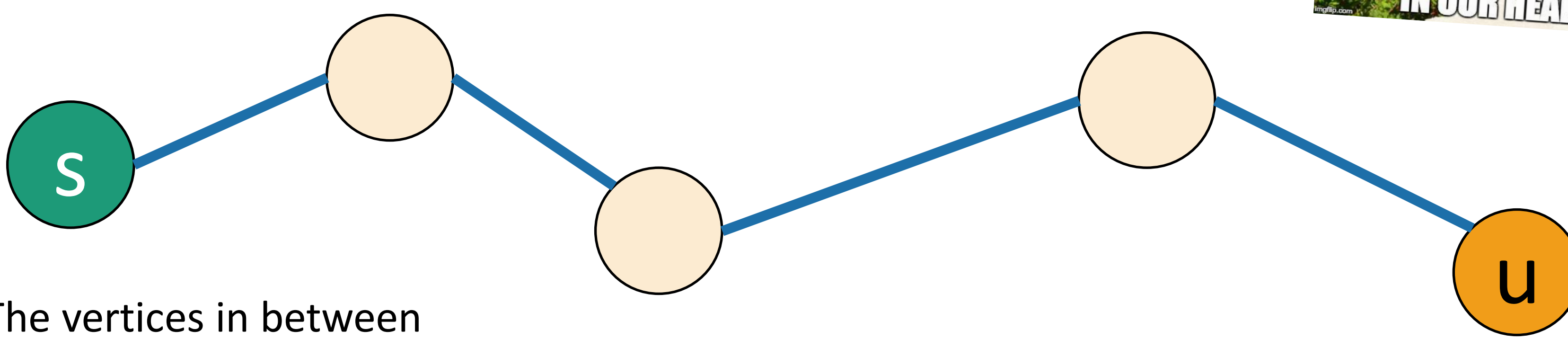
Claim 2

Inductive step

Temporary definition:
v is “good” means that d[v] = d(s,v)

- Want to show that u is good.
- Consider a **true** shortest path from s to u:





The vertices in between are beige because they may or may not be **sure.**

True shortest path.

Claim 2

Inductive step

Temporary definition:

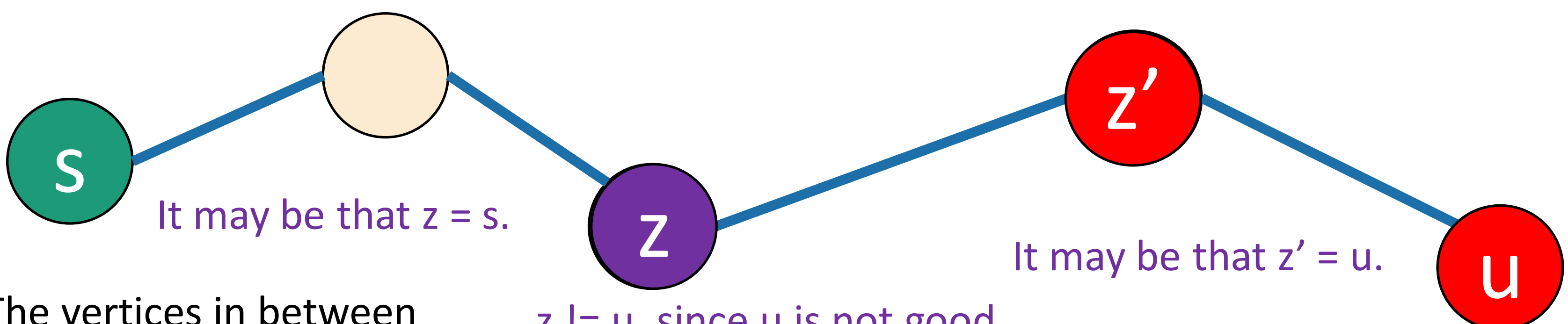
v is "good" means that d[v] = d(s,v)

means good

means not good

"by way of contradiction"

- Want to show that u is good. BWOC, suppose u isn't good.
- Say z is the last good vertex before u.
- z' is the vertex after z.



It may be that z = s.

z != u, since u is not good.

It may be that z' = u.

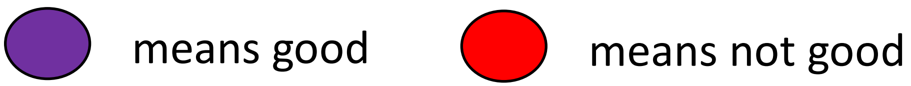
The vertices in between are beige because they may or may not be **sure.**

True shortest path.

Claim 2

Inductive step

Temporary definition:
v is "good" means that d[v] = d(s,v)



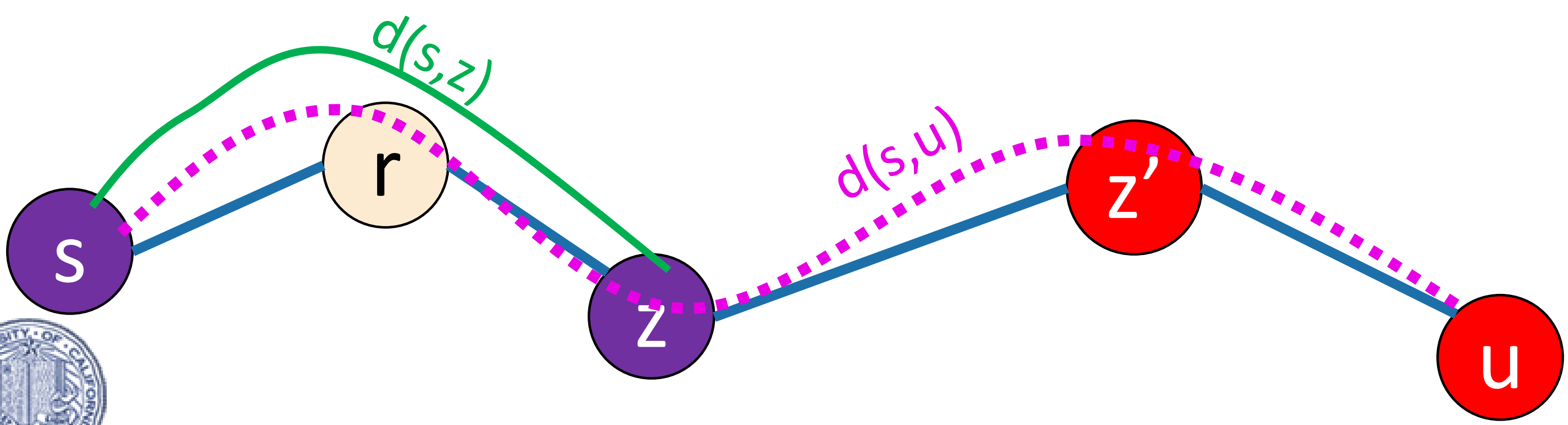
- Want to show that u is good. BWOC, suppose u isn't good.

$$d[z] = d(s,z) \leq d(s,u) \leq d[u]$$

z is good

Subpaths of shortest paths are shortest paths.

AND, also that $d(z,u) \geq 0$, since all of the edge-weights are non-negative!



Claim 2

Inductive step

Temporary definition:

v is "good" means that d[v] = d(s,v)

means good

means not good

- Want to show that u is good. BWOC, suppose u isn't good.

$$d[z] = d(s,z) \leq d(s,u) \leq d[u]$$

z is good

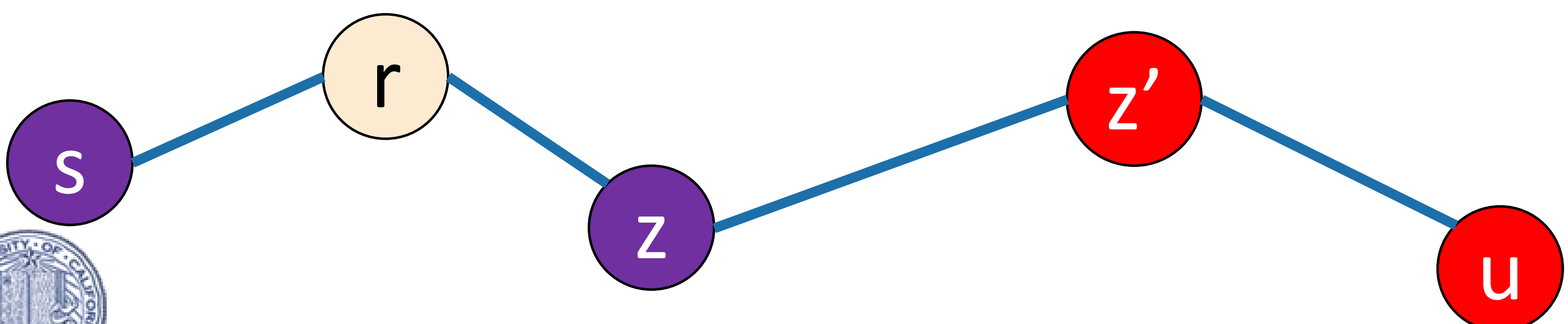
Subpaths of shortest paths are shortest paths.

Claim 1

- If $d[z] = d[u]$, then u is good.
- So $d[z] < d[u]$, so z is **sure.**

But u is not good!

We chose u so that d[u] was smallest of the unsure vertices.



Claim 2

Inductive step

Temporary definition:

v is “good” means that d[v] = d(s,v)

means good

means not good

- Want to show that u is good. BWOC, suppose u isn’t good.

$$d[z] = d(s,z) \leq d(s,u) \leq d[u]$$

z is good

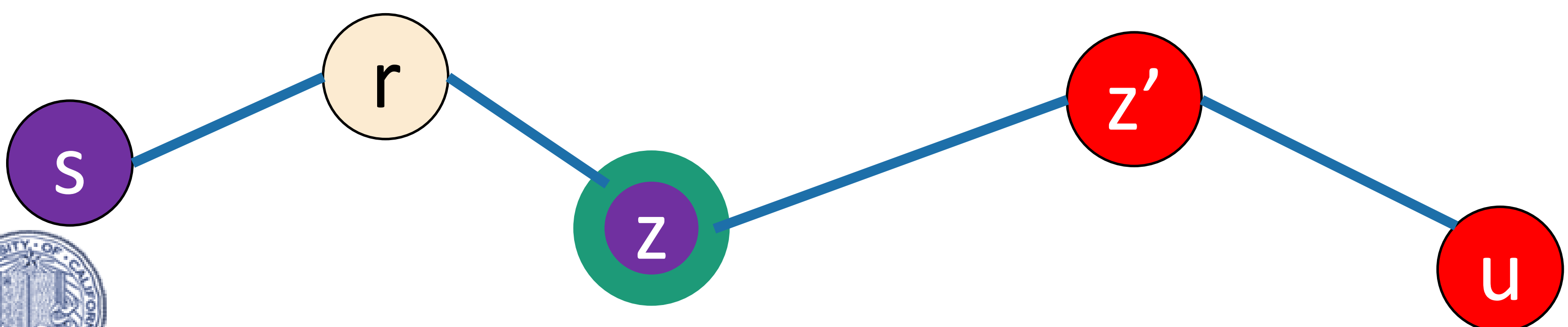
Subpaths of shortest paths are shortest paths.

Claim 1

But u is not good!

- If $d[z] = d[u]$, then u is good.
- So $d[z] < d[u]$, so z is **sure.**

We chose u so that d[u] was smallest of the unsure vertices.



Claim 2

Inductive step

Temporary definition:
v is “good” means that d[v] = d(s,v)

means good   means not good

- Want to show that u is good. BWOC, suppose u isn’t good.
- If z is sure then we’ve already updated z’:

$d[z'] \leftarrow min\{\, d[z'], d[z] + w(z,z')\}$

- $d[z'] \leq d[z] + w(z,z')$ def of update

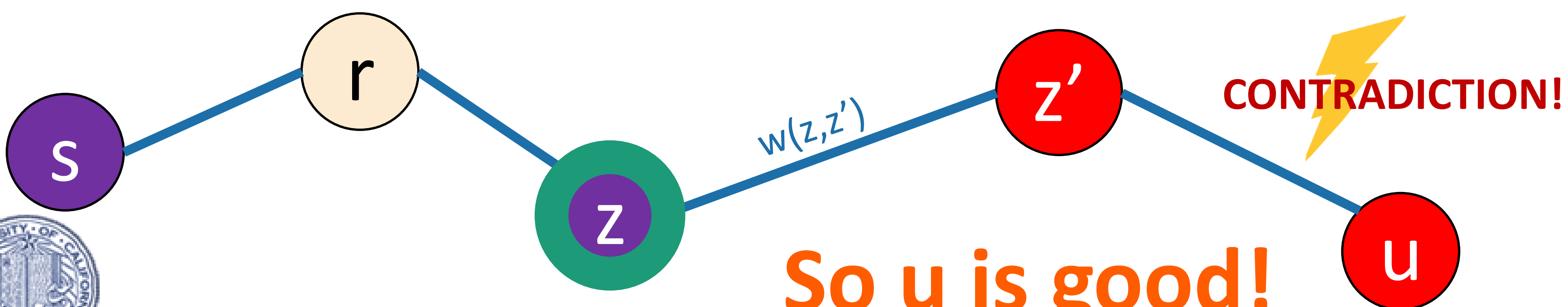
$= d(s,z) + w(z,z')$ By induction when z was added to the sure list it had d(s,z) = d[z]

That is, the value of d[z] when z was marked sure…

$= d(s,z')$ sub-paths of shortest paths are shortest paths

$\leq d[z']$ Claim 1

So d(s,z’) = d[z’] and so z’ is good.



So u is good!

Claim 2

Back to this slide

When a vertex u is marked sure, d[u] = d(s,u)

- Inductive Hypothesis:
  - When we mark the t'th vertex v as sure, d[v] = dist(s,v).
- Base case:
  - The first vertex marked **sure** is s, and d[s] = d(s,s) = 0.
- Inductive step:
  - Suppose that we are about to add u to the **sure** list.
  - That is, we picked u in the first line here:

> - Pick the **not-sure** node u with the smallest estimate **d[u].**
> - Update all u's neighbors v:
>   - d[v] ← min(d[v] , d[u] + edgeWeight(u,v))
> - Mark u as **sure**.
> - Repeat

  - Assume by induction that every v already marked **sure** has d[v] = d(s,v).
  - Want to show that d[u] = d(s,u).

Why does this work?

Now back to this slide

- **Theorem**:
  - Run Dijkstra on G =(V,E) starting from s.
  - At the end of the algorithm, the estimate **d[v]** is the actual distance d(s,v).

- Proof outline:
  - **Claim 1**: For all v, **d[v] $\geq$ d(s,v).**
  - **Claim 2**: When a vertex is marked **sure**, **d[v] = d(s,v).**

- **Claims 1 and 2** imply the **theorem.**

What have we learned?

- Dijkstra's algorithm finds shortest paths in weighted graphs with non-negative edge weights.
- Along the way, it constructs a nice tree.
  - We could post this tree in COB1!
  - Then people would know how to get places quickly.