Shortest Paths

Data Structures and Parallelism

Announcements

Parallelism exercises due today

“One” more exercise out this afternoon.

We’ve written two exercises 12A and 12B.

You can either

1. Submit only one of A and B (whichever one you prefer)
2. Or use a token and do both (you get the higher of the two scores).

Because we’re at the end of the quarter, you won’t be able to use a token to redo exercise 12.

Both are useful for studying for the final, you should at least look at both.

Announcements

P3 checkpoint 2 Wednesday.

Using tokens to redo exercises:
Redone exercises will be due next Tuesday (Aug 14) at 11:59 PM.

We'll have a form to tell us

- How many tokens you're using for P3 late days and
- Which exercises you're redoing.

Have to decide by Tuesday Aug. 14.

Redone parallelism exercises will be submitted on gitlab.

Others submitted on gradescope.

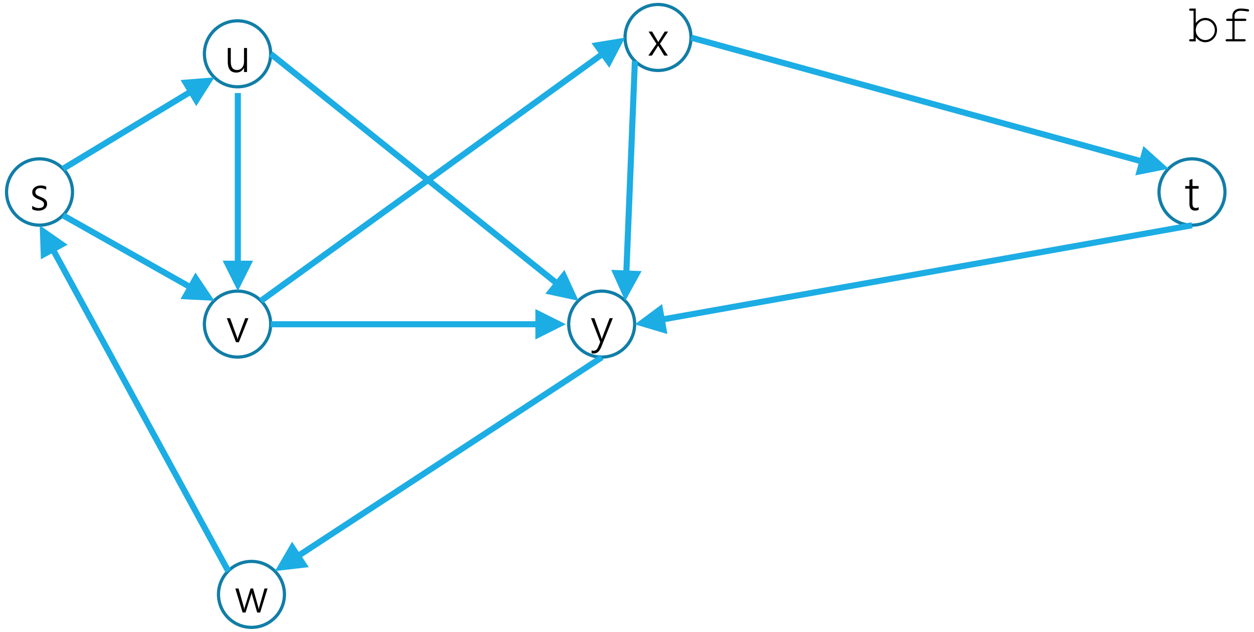
Warm Up

Run Breadth First Search on this graph starting from s.

What order are vertices placed on the queue?

When processing a vertex insert neighbors in alphabetical order.

In a directed graph, BFS only follows an edge in the direction it points.




```
bfs(graph)
   toVisit.enqueue(first vertex)
   mark first vertex as visited
   while(toVisit is not empty)
      current = toVisit.dequeue()
      for (V : current.outneighbors())
         if (v is not visited)
            toVisit.enqueue(v)
           mark v as visited
      finished.add(current)
```

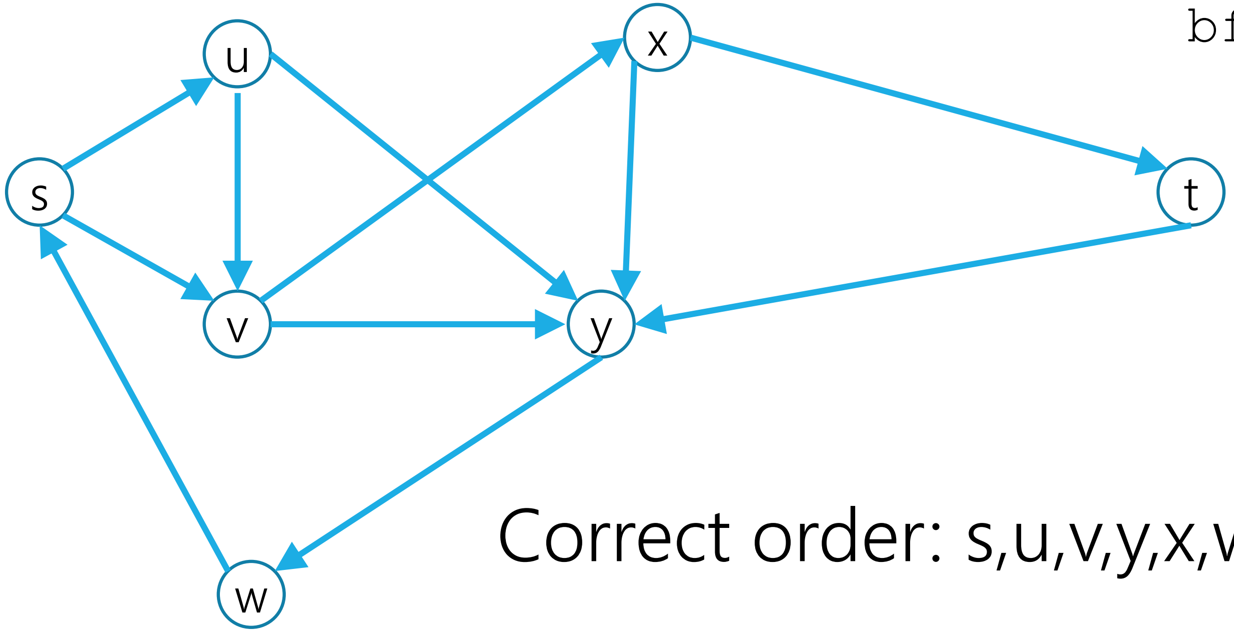
Warm Up

Run Breadth First Search on this graph starting from s.

What order are vertices placed on the queue?

When processing a vertex insert neighbors in alphabetical order.

In a directed graph, BFS only follows an edge in the direction it points.

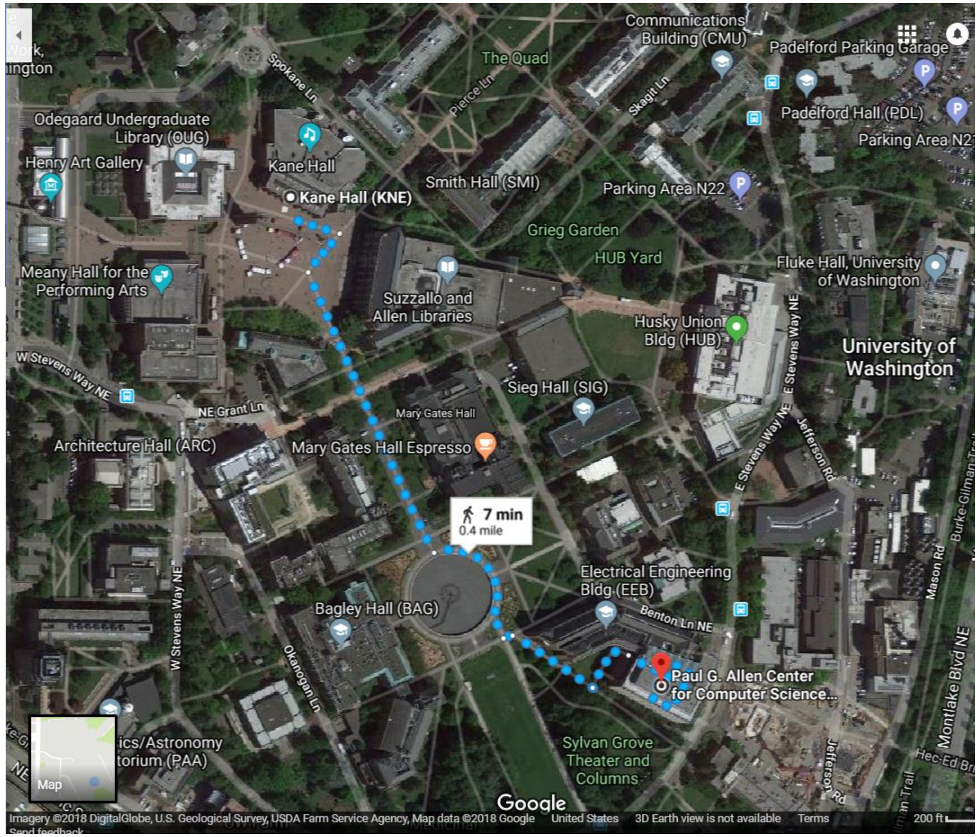



```
bfs(graph)
    toVisit.enqueue(first vertex)
    mark first vertex as visited
    while(toVisit is not empty)
        current = toVisit.dequeue()
        for (V : current.outneighbors())
            if (v is not visited)
                toVisit.enqueue(v)
              mark v as visited
        finished.add(current)
```

Correct order: s,u,v,y,x,w,t

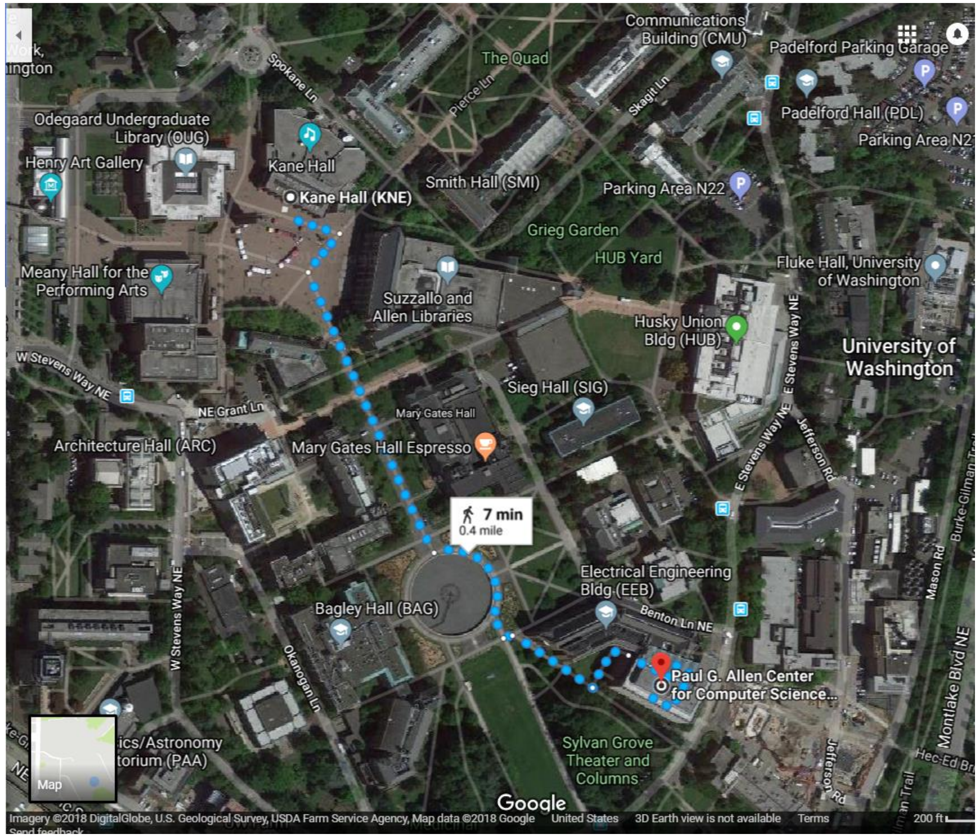
Shortest Paths

How does Google Maps figure out this is the fastest way to get from Kane Hall to CSE?

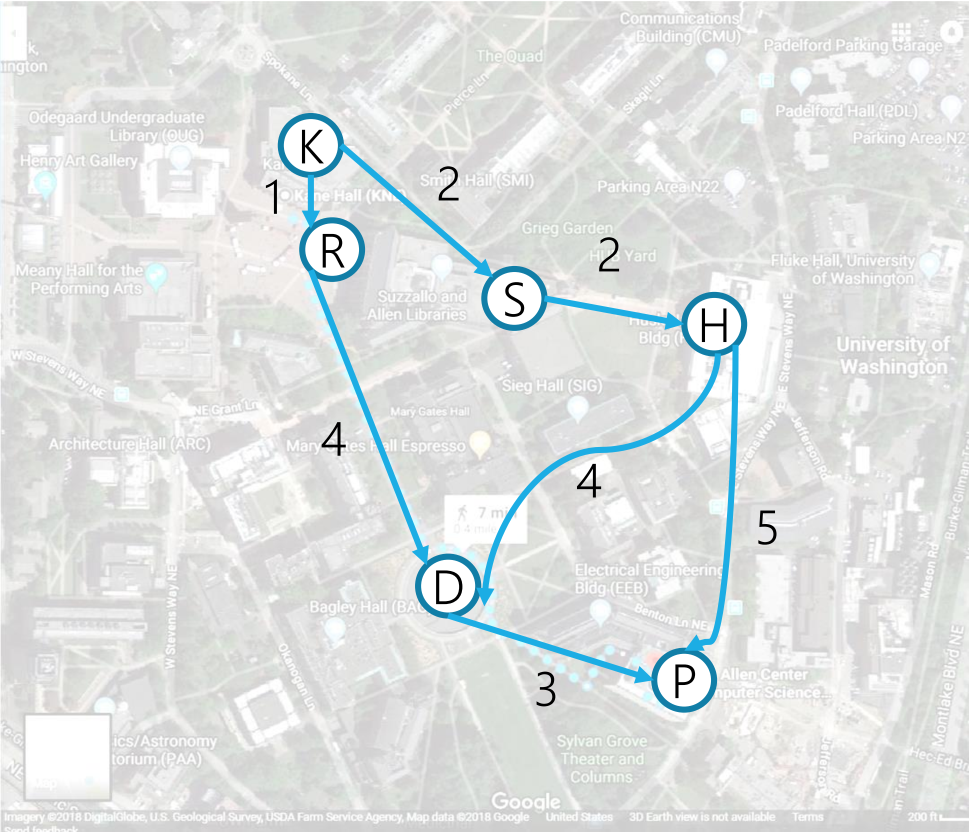


Representing Maps as Graphs

How do we represent a map as a graph? What are the vertices and edges?



Representing Maps as Graphs

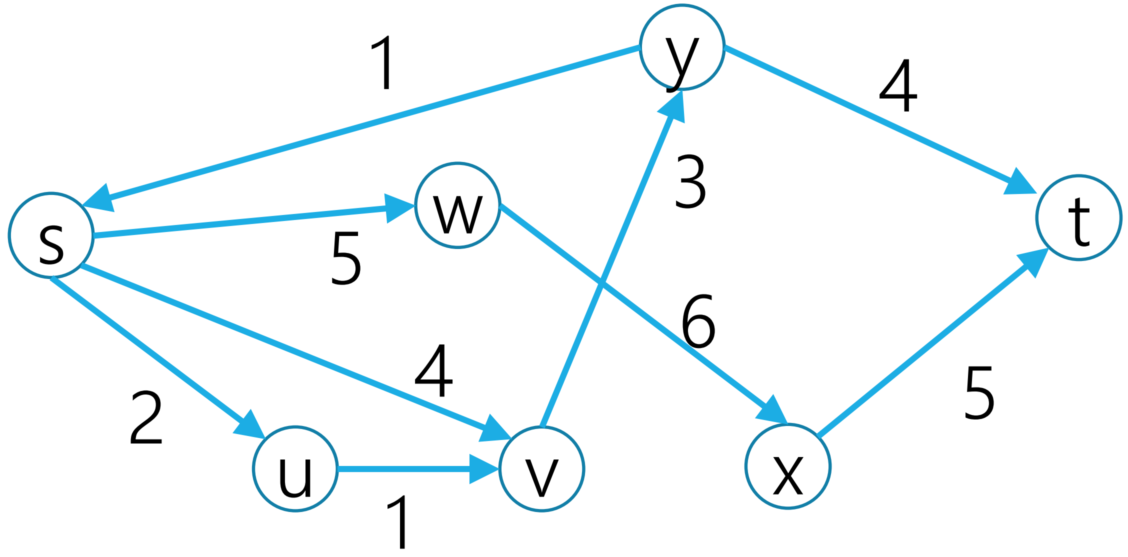


Shortest Paths

The **length** of a path is the sum of the edge weights on that path.

Shortest Path Problem

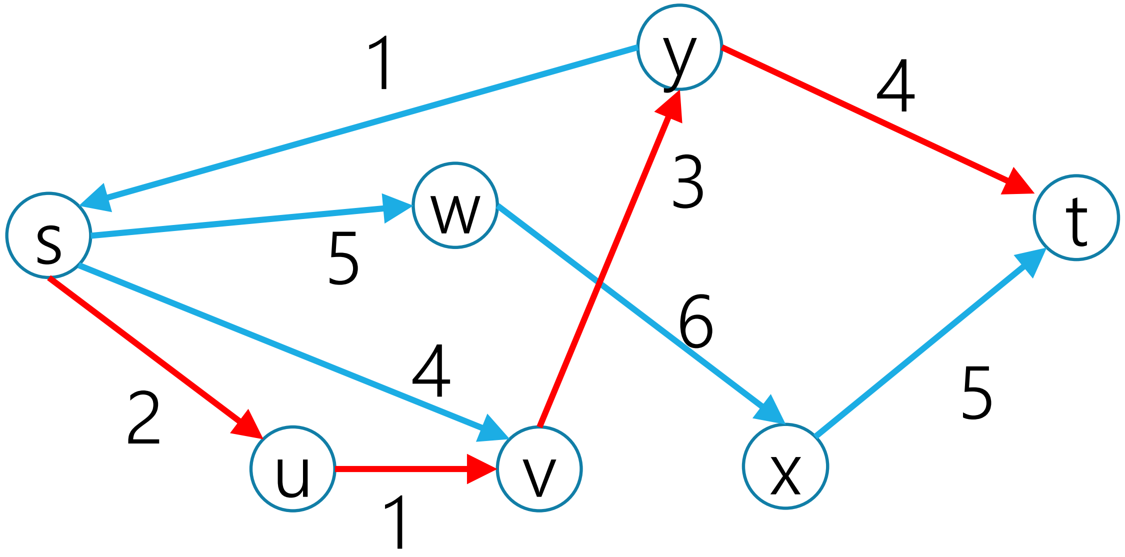
Given a directed graph and vertices s and t
Find: the shortest path from s to t.



Shortest Paths

The **length** of a path is the sum of the edge weights on that path.

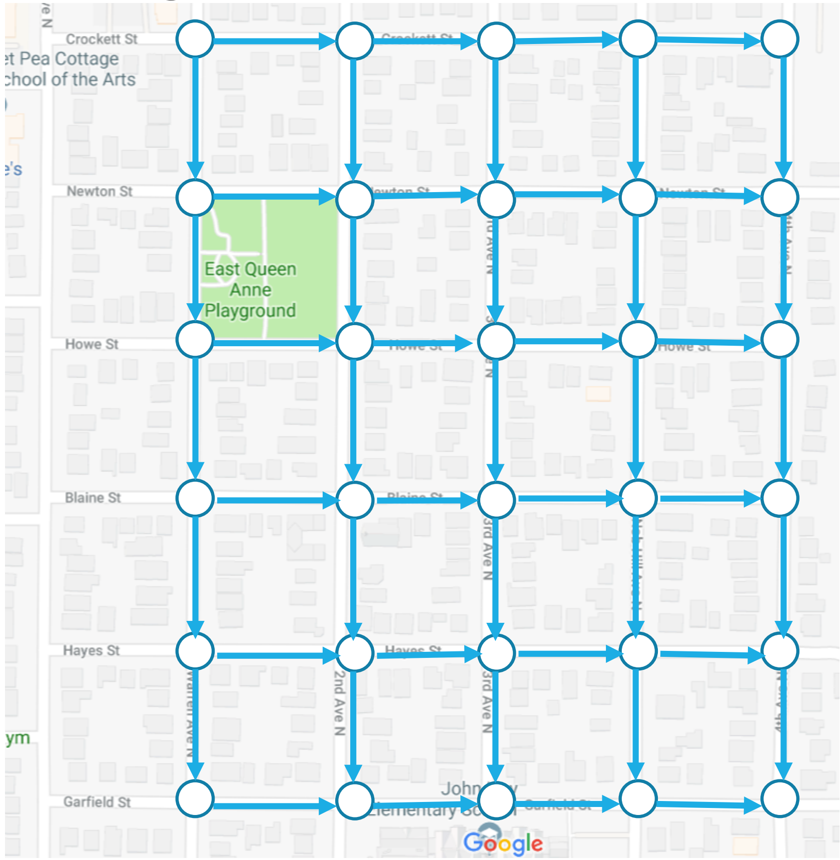
Shortest Path Problem
Given a directed graph and vertices s and t Find: the shortest path from s to t.



Unweighted graphs

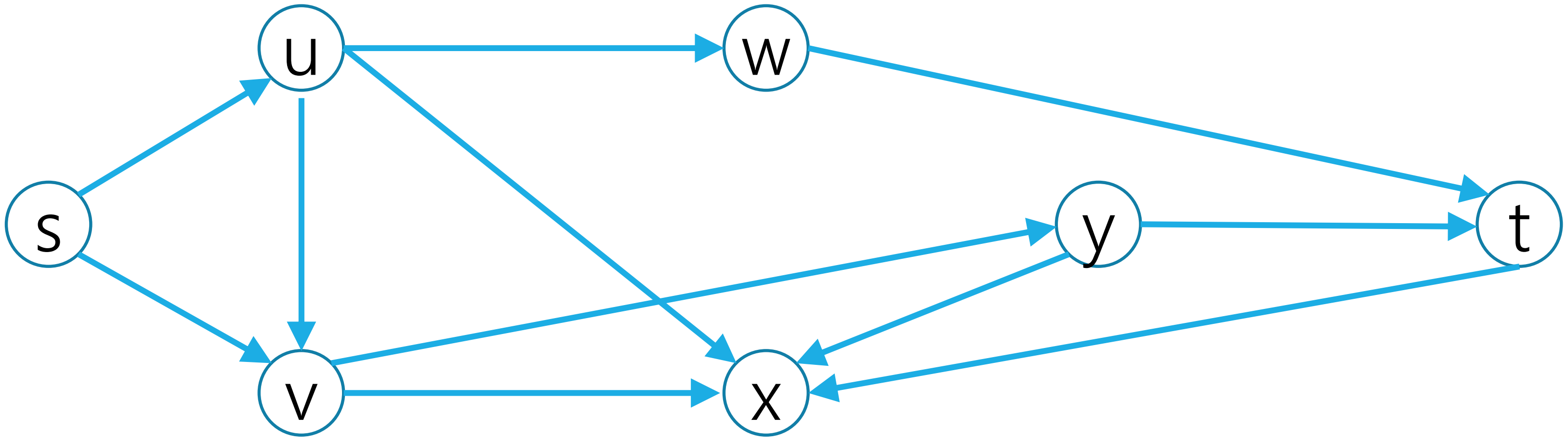
Let’s start with a simpler version: the edges are all the same weight

If the graph is **unweighted**, how do we find a shortest paths?



Unweighted Graphs

If the graph is unweighted, how do we find a shortest paths?



What's the shortest path from s to s?

- Well....we're already there.

What's the shortest path from s to u or v?

- Just go on the edge from s

From s to w,x, or y?

- Can't get there directly from s, for length 2 path, have to go through u or v.

Unweighted Graphs: Key Idea

To find the set of vertices at distance k, just find the set of vertices at distance k-1, and check for outgoing edge to an undiscovered vertex.

Do we already know an algorithm that does something like that?

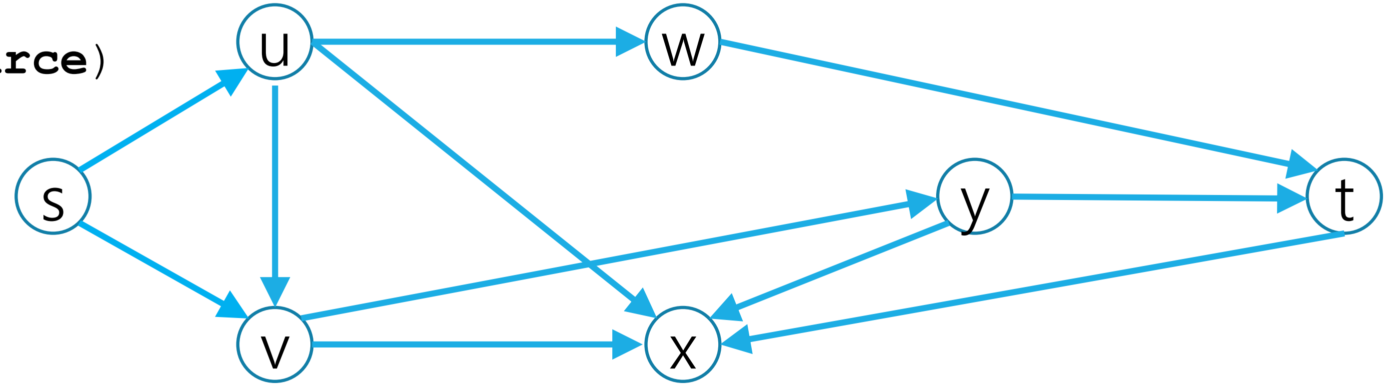
Yes! BFS!

```
bfsShortestPaths(graph G, vertex source)
   toVisit.enqueue(source)
   source.dist = 0
   mark source as visited
   while(toVisit is not empty){
      current = toVisit.dequeue()
      for (v : current.outNeighbors()){
         if (v is not yet visited){
            v.distance = current.distance + 1
            v.predecessor = current
            toVisit.enqueue(v)
            mark v as visited
         }
      }
   }
```

Unweighted Graphs

Use BFS to find shortest paths in this graph.

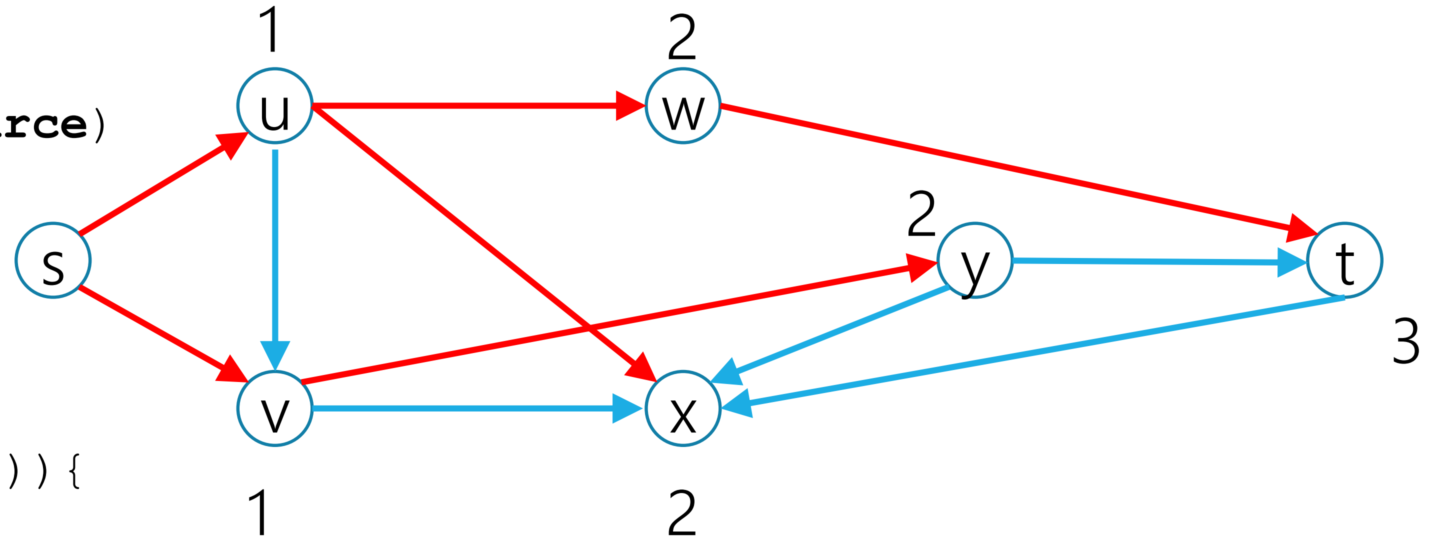
```
bfsShortestPaths(graph G, vertex source)
   toVisit.enqueue(source)
   source.dist = 0
   mark source as visited
   while(toVisit is not empty){
      current = toVisit.dequeue()
      for (v : current.outNeighbors()){
         if (v is not yet visited){
            v.distance = current.distance + 1
            v.predecessor = current
            toVisit.enqueue(v)
            mark v as visited
         }
      }
   }
```



Unweighted Graphs

Use BFS to find shortest paths in this graph.

```
bfsShortestPaths(graph G, vertex source)
   toVisit.enqueue(source)
   source.dist = 0
   mark source as visited
   while(toVisit is not empty){
      current = toVisit.dequeue()
      for (v : current.outNeighbors()){
         if (v is not yet visited){
            v.distance = current.distance + 1
            v.predecessor = current
            toVisit.enqueue(v)
            mark v as visited
         }
      }
   }
```



What about the target vertex?

Shortest Path Problem

Given a directed graph and vertices s and t
Find: the shortest path from s to t.

BFS didn't mention a target vertex…
It actually finds the shortest path from s to every other vertex.

If you know your target, you can stop the algorithm early, when the target is removed from the queue.

Weighted Graphs

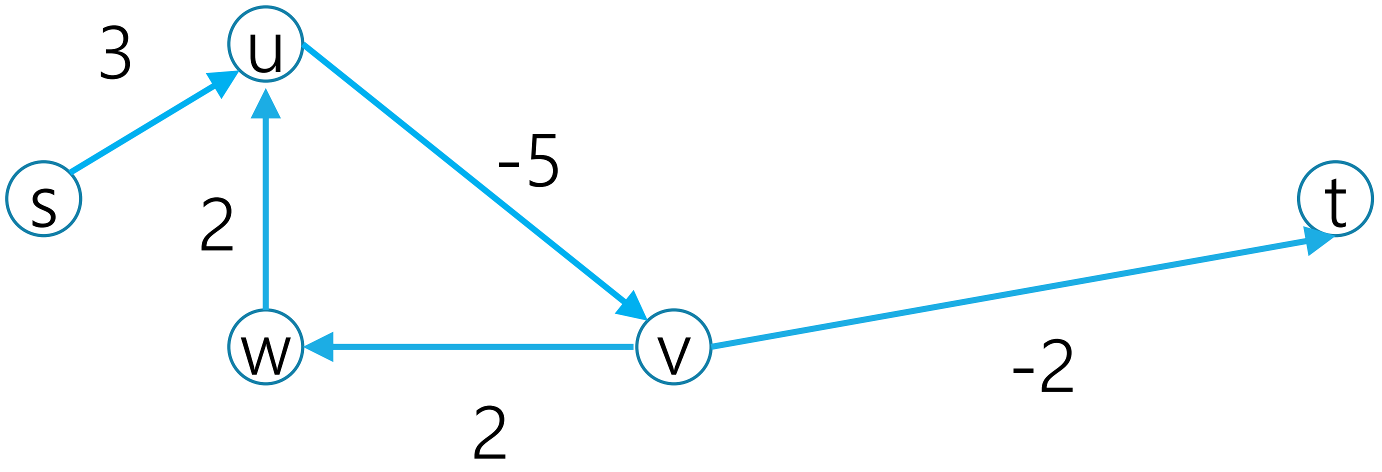
Each edge should represent the “time” or “distance” from one vertex to another.

Sometimes those aren’t uniform, so we put a weight on each edge to record that number.

The length (or “**weight**” or “**cost**”) of a path in a weighted graph is the sum of the weights along that path.

Negative Edge Weights

What's the shortest way to get from s to t?



s, u,v,w, u,v,w, u,v,w, …

There is no shortest way. You can always go around u,v,w once more.

If there's a **negative weight cycle** shortest paths are **undefined.**

Negative Edge Weights

If there are negative edge weights, but no negative weight cycle, shortest paths are still defined.

For today we'll assume all of the weights are positive

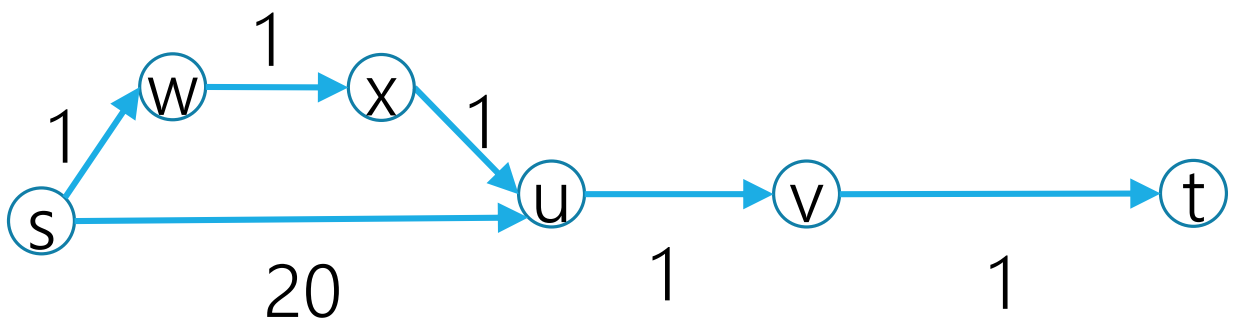
- For GoogleMaps that definitely makes sense.
- Sometimes negative weights make sense.
- **Today's algorithm doesn't work for those graphs**
- There are other algorithms that do work (ask Robbie later)

In section, you'll see why negative edge weights might be useful.

Weighted Graphs: Take 1

BFS works if the graph is unweighted.

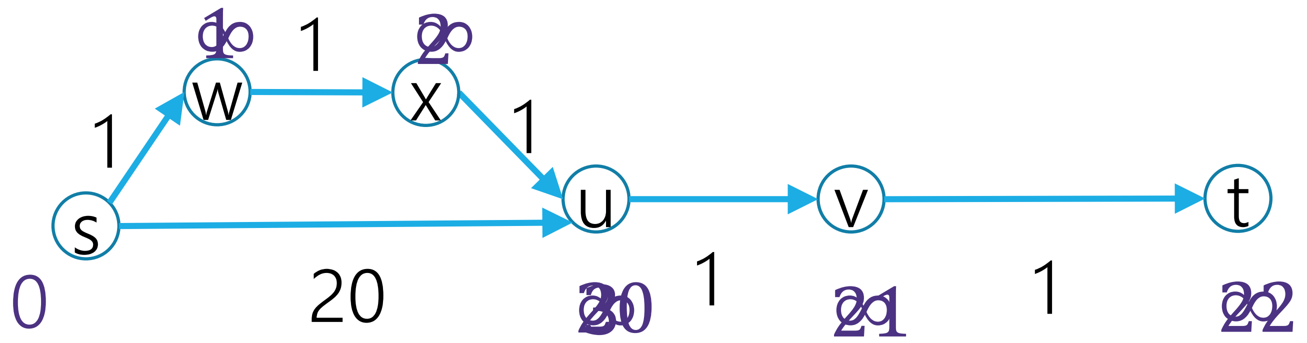
Maybe it just works for weighted graphs too?



Weighted Graphs: Take 1

BFS works if the graph is unweighted.

Maybe it just works for weighted graphs too?



What went wrong?
When we found a shorter path from s to u,
we needed to update the distance to v
but BFS doesn't do that.

Weighted Graphs: Take 2

Reduction (informally)

Using an algorithm for Problem B to solve Problem A.

You already do this all the time.

In P1, you reduced implementing a hashset to implementing a hashmap.

They appeared in exercise 7.

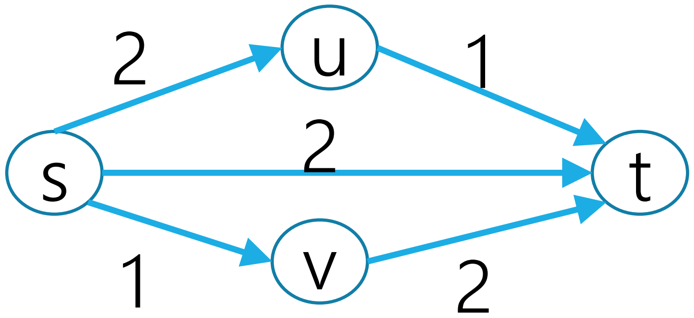
Any time you use a library, you're reducing your problem to the one the library solves.

Can we reduce finding shortest paths on weighted graphs to finding them on unweighted graphs?

Weighted Graphs Take 2

Given a weighted graph, how do we turn it into an unweighted one without messing up the path lengths?

Weighted Graphs: A Reduction



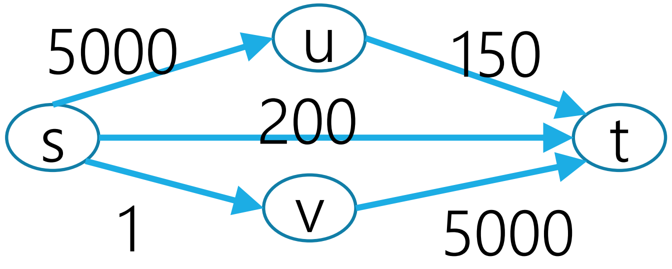
Transform Input

Unweighted Shortest Paths

Transform Output

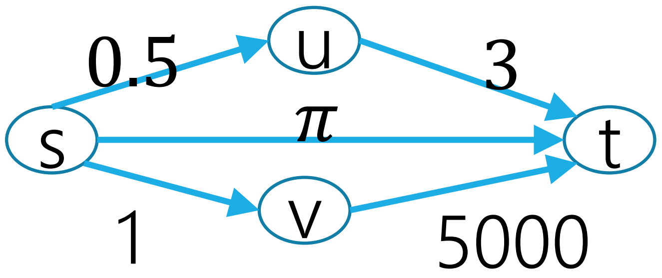
Weighted Graphs: A Reduction

What is the running time of our reduction on this graph?



O(|V|+|E|) of the modified graph, which is…slow.

Does our reduction even work on this graph?



Ummm….

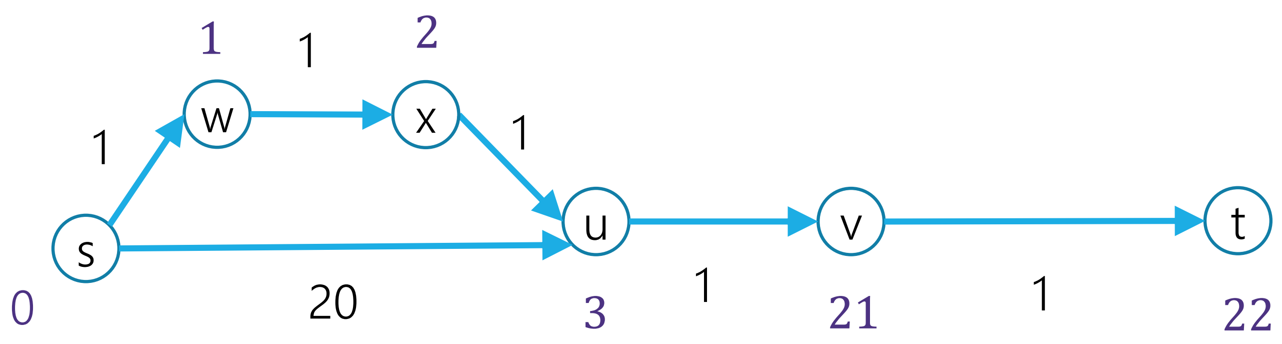
Tl;dr: If your graph's weights are all small positive integers, this reduction might work great.
Otherwise we probably need a new idea.

Weighted Graphs: Take 3

So we can't just do a reduction.

Instead figure out why BFS worked in the unweighted case, try to make the same thing happen in the weighted case.

How did we avoid this problem:



Weighted Graphs: Take 3

In BFS When we used a vertex u to update shortest paths we already knew the exact shortest path to u.

So we never ran into the update problem

If we process the vertices in order of distance from s, we have a chance.

Weighted Graphs: Take 3

Goal: Process the vertices in order of distance from s

Idea:

Have a set of vertices that are “known”

- (we know at least one path from s to them).

Record an estimated distance

- (the best way we know to get to each vertex).

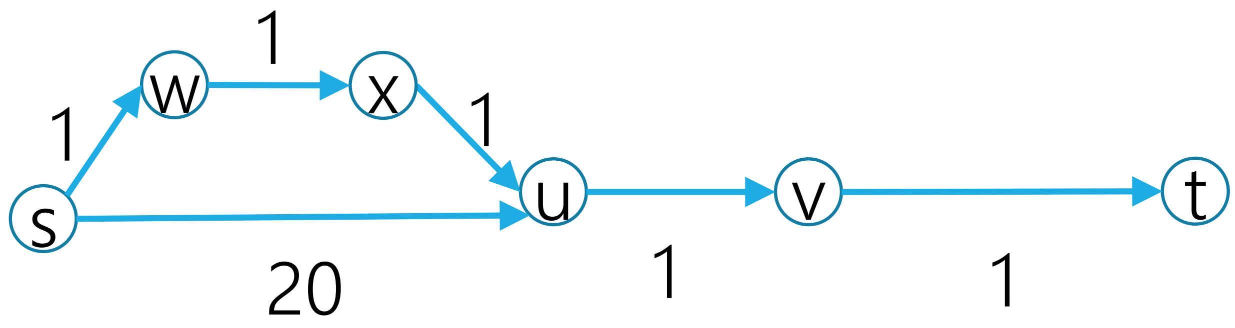
If we process only the vertex closest in estimated distance, we won’t ever find a shorter path to a processed vertex.

- This statement is the key to proving correctness.
- It’s nice if you want to practice induction/understand the algorithm better.

Dijkstra’s Algorithm

```
Dijkstra(Graph G, Vertex source)
   initialize distances to ∞
   mark source as distance 0
   mark all vertices unprocessed
   while(there are unprocessed vertices){
       let u be the closest unprocessed vertex
       foreach(edge (u,v) leaving u){
           if(u.dist+weight(u,v) < v.dist){
               v.dist = u.dist+weight(u,v)
               v.predecessor = u
           }
       }
      mark u as processed
   }
```

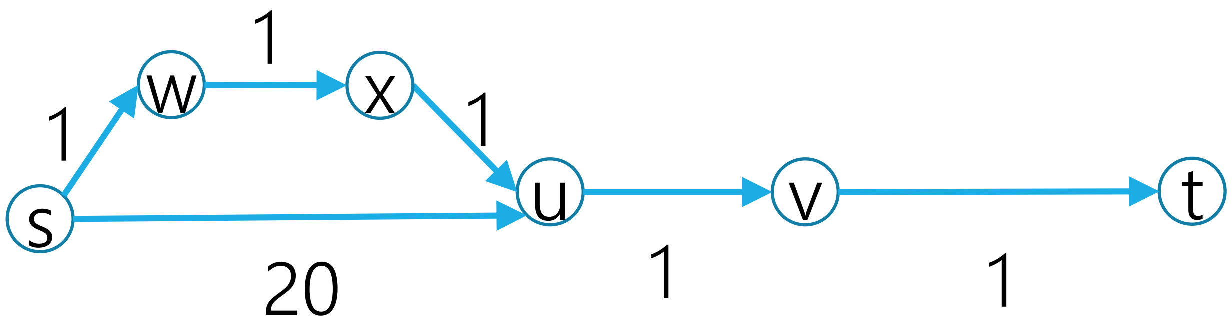
Vertex	Distance	Predecessor	Processed
s			
w			
x			
u			
v			
t			



Dijkstra’s Algorithm

```
Dijkstra(Graph G, Vertex source)
   initialize distances to ∞
   mark source as distance 0
   mark all vertices unprocessed
   while(there are unprocessed vertices){
       let u be the closest unprocessed vertex
       foreach(edge (u,v) leaving u){
           if(u.dist+weight(u,v) < v.dist){
               v.dist = u.dist+weight(u,v)
               v.predecessor = u
           }
       }
      mark u as processed
   }
```

Vertex	Distance	Predecessor	Processed
s	0	--	Yes
w	1	s	Yes
x	2	w	Yes
u	~~20~~ 3	~~s~~ x	Yes
v	4	u	Yes
t	5	v	Yes



Implementation Details

One of those lines of pseudocode was a little sketchy

```
> let u be the closest unprocessed vertex
```

What ADT have we talked about that might work here?

Minimum Priority Queues!

Making Minimum Priority Queues Work

They won't quite work "out of the box".

We don't have an update priority method. Can we add one?

- Percolate up!

To percolate u's entry in the heap up we'll have to get to it.

- Each vertex need pointer to where it appears in the priority queue
- I'm going to ignore this point for the rest of the lecture.

Min Priority Queue ADT
state Set of comparable values - Ordered by "priority" **behavior** **peek()** – find the element with the smallest priority **insert(value)** – add new element to collection **removeMin()** – returns and removes element with the smallest priority **DecreaseKey(e, p)** – decreases priority of element e down to p.

Running Time Analysis

```
Dijkstra(Graph G, Vertex source)
    initialize distances to ∞, source.dist to 0
    mark all vertices unprocessed
    initialize MPQ as a Min Priority Queue
    add source at priority 0
    while(MPQ is not empty){
        u = MPQ.removeMin()
        foreach(edge (u,v) leaving u){
            if(u.dist+weight(u,v) < v.dist){
                if(v.dist == ∞ ) //if v not in MPQ
                    MPQ.insert(v, u.dist+weight(u,v))
                else
                    MPQ.decreaseKey(v, u.dist+weight(u,v))
                v.dist = u.dist+weight(u,v)
                v.predecessor = u
            }
        }
        mark u as processed
    }
```

Optional Content
More Graph Applications

Another Application of Shortest Paths

Shortest path algorithms are obviously useful for GoogleMaps.

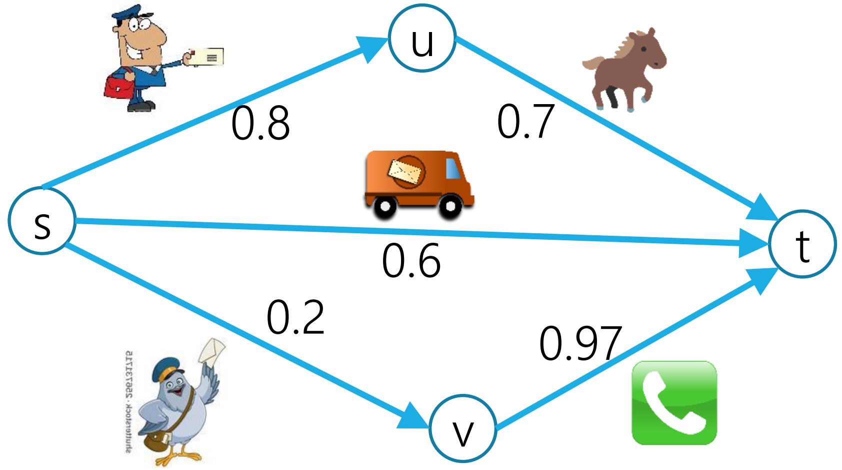
The wonderful thing about graphs is they can encode **arbitrary** relationships among objects.

I don't care if you remember all the details.

I just want you to see that these algorithms have non-obvious applications.

Another Application of Shortest Paths

I have a message I need to get from point s to point t.
But the connections are unreliable.
What path should I send the message along so it has the best chance of arriving?



Maximum Probability Path

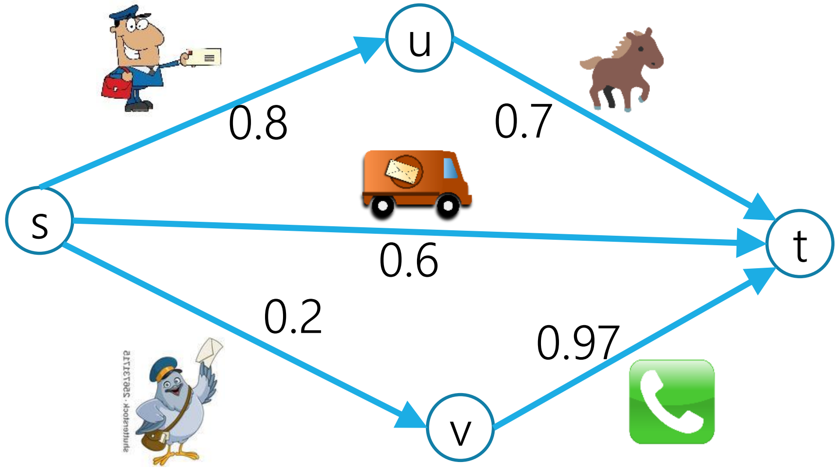
Given: a directed graph G, where each edge weight is the probability of successfully transmitting a message across that edge
Find: the path from s to t with maximum probability of message transmission

Another Application of Shortest Paths

Let each edge's weight be the probability a message is sent successfully across the edge.

What's the probability we get our message all the way across a path?

- It's the product of the edge weights.

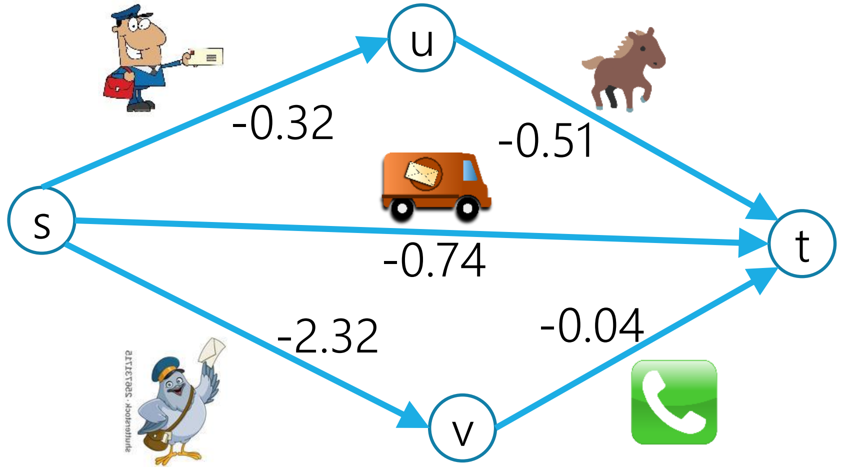


We only know how to handle sums of edge weights.

Is there a way to turn products into sums?

$\log(ab) = \log a + \log b$

Another Application of Shortest Paths



We've still got two problems.

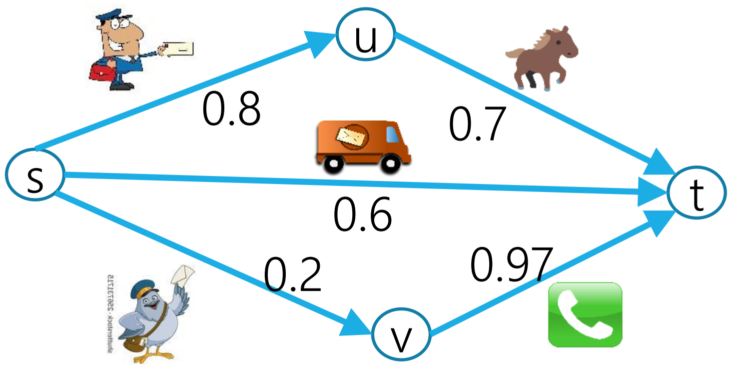
1. When we take logs, our edge weights become negative.

2. We want the *maximum* probability of success, but that's the longest path not the shortest one.

Multiplying all edge weights by negative one fixes both problems at once!

We **reduced** the maximum probability path problem to a shortest path problem by taking $-\log()$ of each edge weight.

Maximum Probability Path Reduction



Transform Input

Weighted Shortest Paths

Transform Output