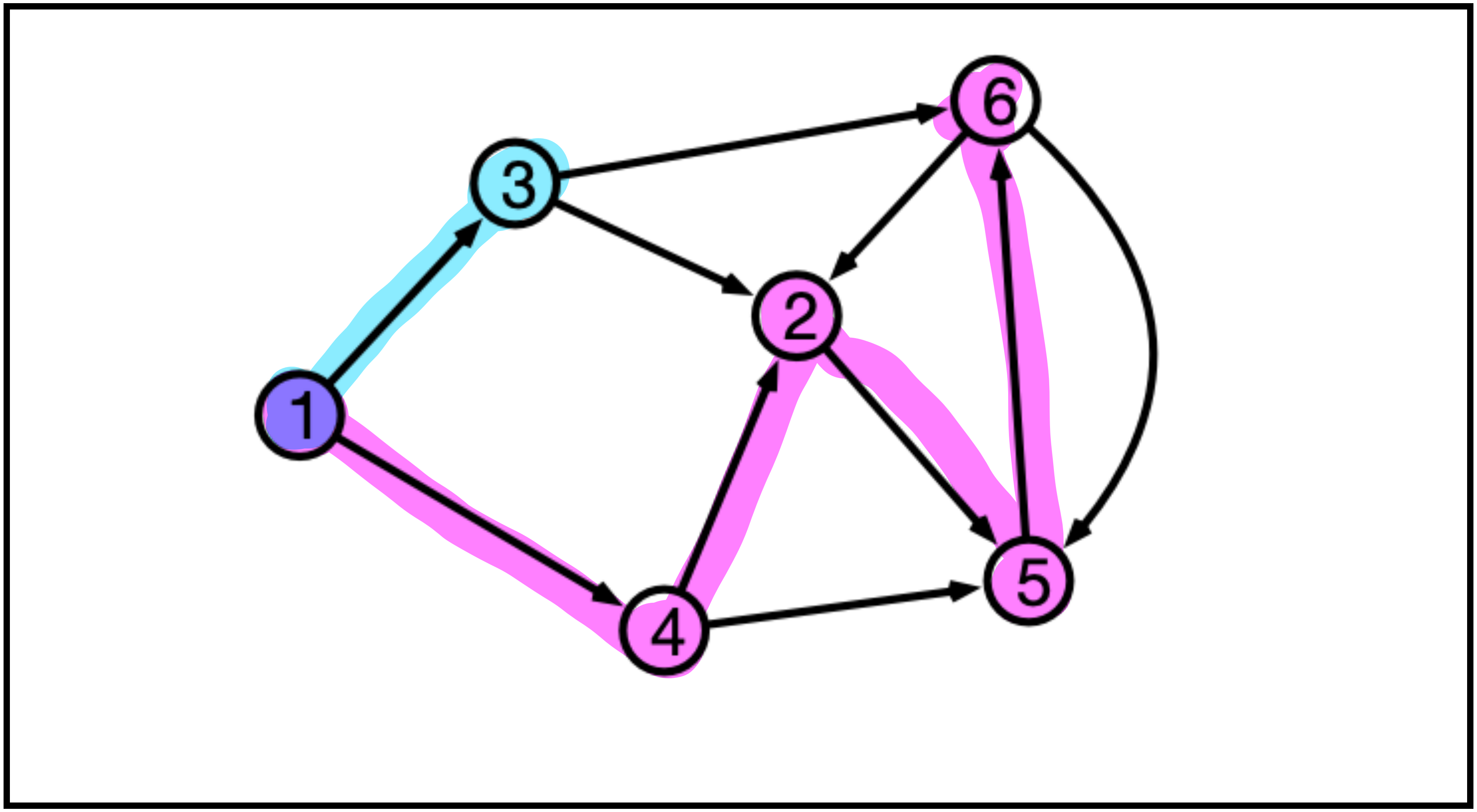
# Lecture 27: Shortest Paths, Revisited

COSC 311 *Algorithms*, Fall 2022

## Announcements

1. HW 05 Due this Friday
2. Midterm 2 Wednesday 11/16
    - in class
    - focus on material since last midterm up to this Friday
    - algorithmic paradigms:
        - divide and conquer
        - greedy
        - dynamic programming
        - network flow?
    - mixture of computational and algorithm design questions
    - details forthcoming

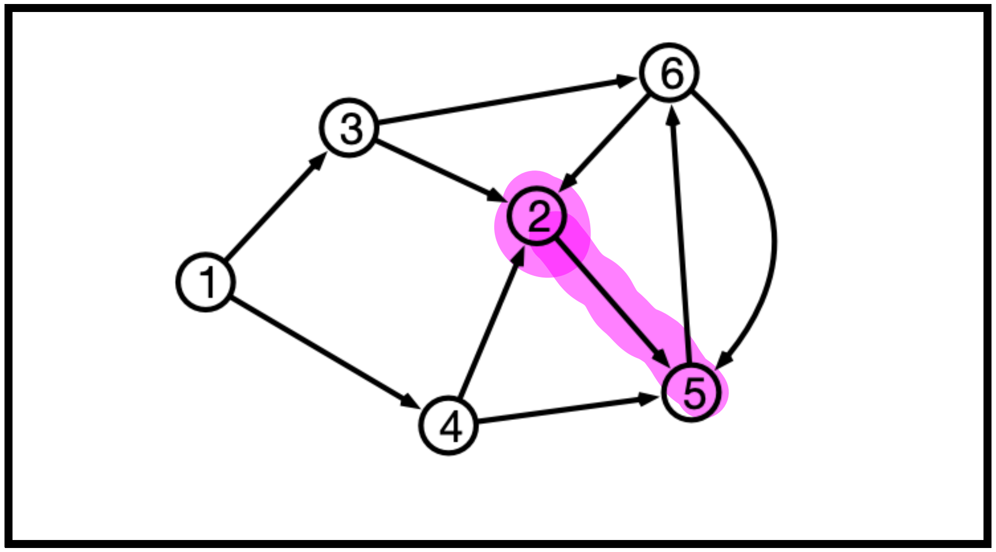
# Directed Graphs and Paths



## Representing Directed Graphs

### Adjacency List

- *v*'s neighbors are *outgoing* neighbors



| | |
|---|---|
| 1 : | 3, 4 |
| 2 : | 5 |
| 3 : | 2, 6 |
| 4 : | 2, 5 |
| 5 : | 6 |
| 6 : | 2, 5 |

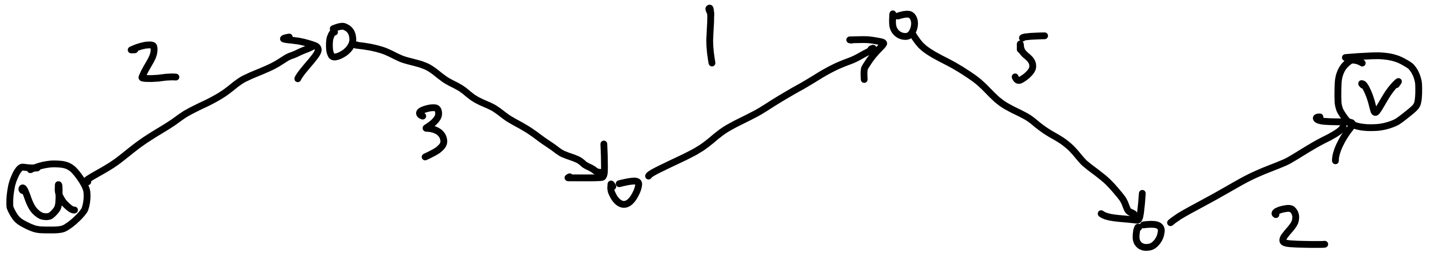
# Single Source Shortest Paths

**Input:**

- (Directed) graph $G = (V, E)$, edge weights $w$
- Starting vertex $u$

**Output:**

- $d(v)$ = distance from $u$ to $v$ for every vertex $v$
  - distance is length of shortest *directed* path from $u$ to $v$
  - length is sum of edge weights of path



$w(p) = 2+3+1+5+2 = 13$

## Previous Algorithms

1. Breadth-first Search (BFS)
    - solves SSSP when all edge weights are 1
2. Dijkstra's Algorithm
    - solves SSSP when all edge weights are $\geq 0$

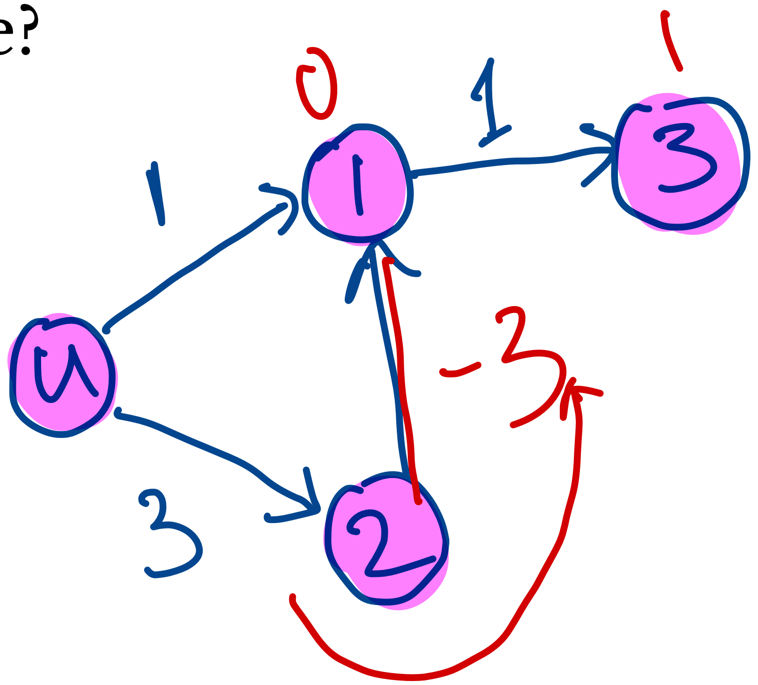
## Previous Algorithms

1. Breadth-first Search (BFS)
    - solves SSSP when all edge weights are 1
2. Dijkstra's Algorithm
    - solves SSSP when all edge weights are $\geq 0$

**Question.** What if edge weights can be negative?

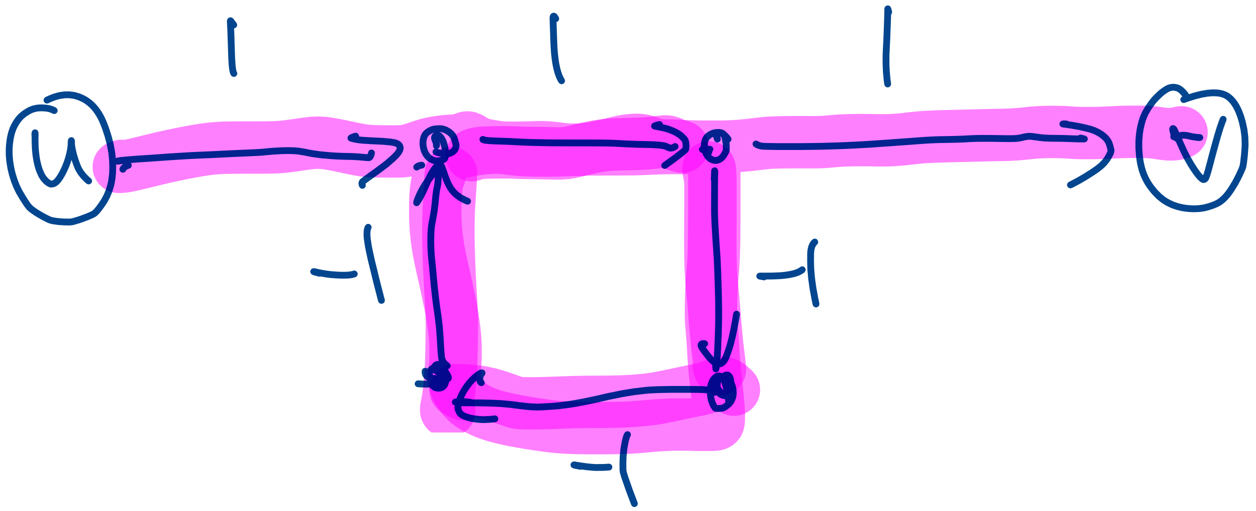
## Example 1

**Question 1.** Why can Dijkstra fail when edge weights can be negative?



## Example 2

**Question 2.** Why might shortest paths not exist when edge weights can be negative?



## Assumption

**Assume.** $G$ does not contain any *negative weight cycles*.

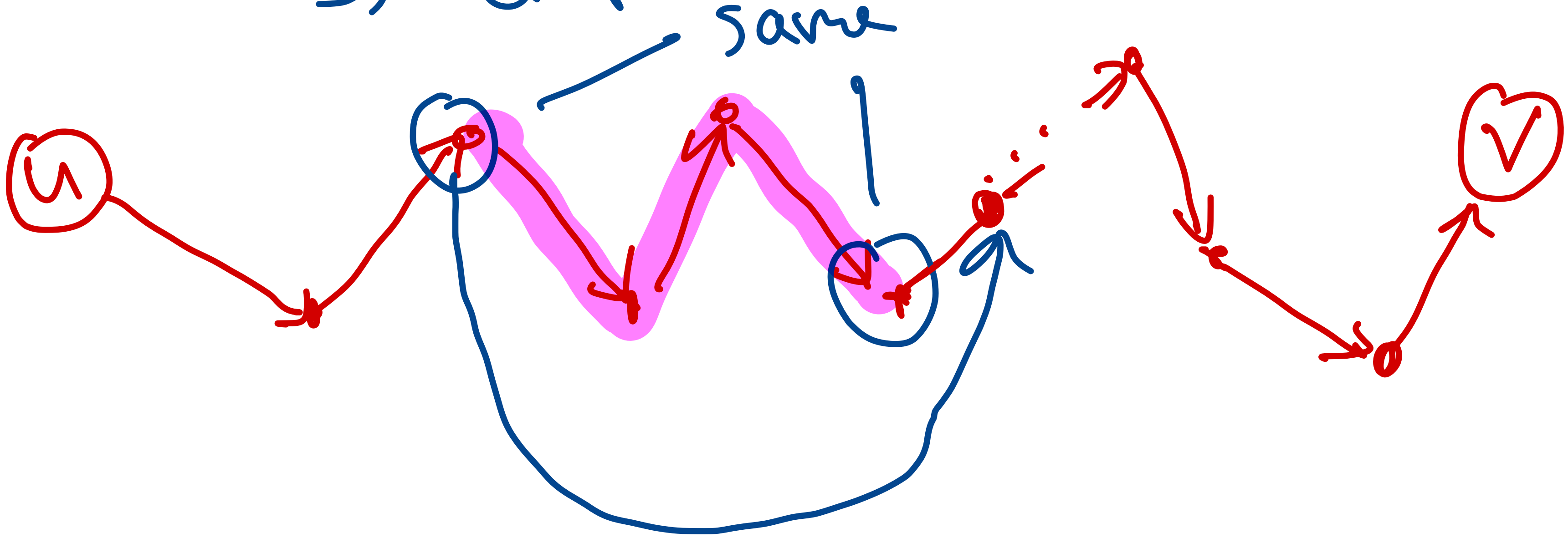
## Assumption

**Assume.** $G$ does not contain any *negative weight cycles.*

**Claim.** $G$ a graph with $n$ vertices, $u, v$ vertices in $G$. If $G$ does not contain negative weight cycles, then the shortest (weighted) path from $u$ to $v$ contains at most $n - 1$ edges.

*Why?* $\geq n$ edges, $\geq n+1$ vertices

$\Rightarrow$ duplicate



$\ell$ edges,

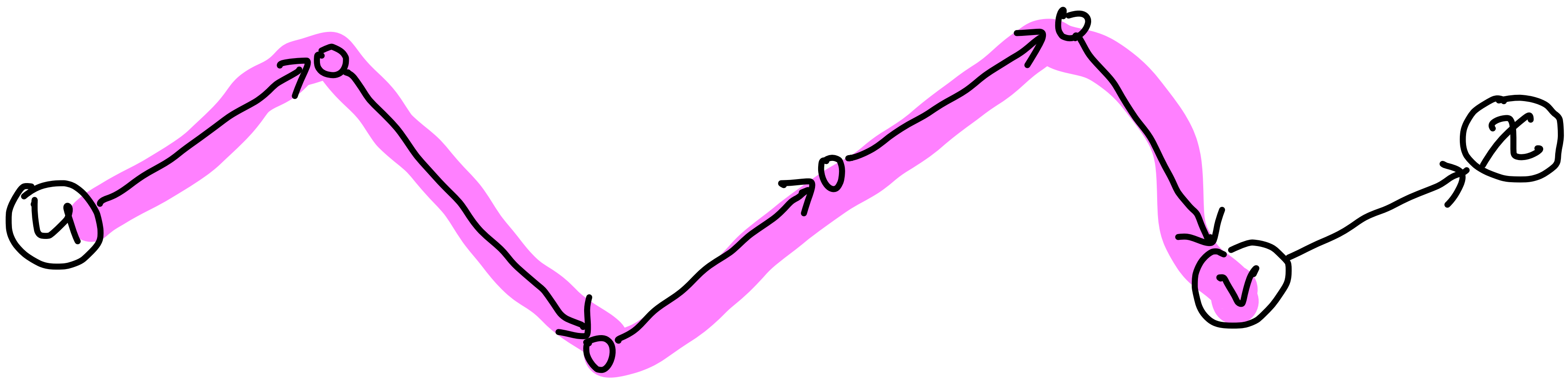
how many vertices? $\ell+1$

# Shortest Paths Again

Suppose shortest path from $u$ to $x$ contains $j$ hops.

- $v$ is $x$'s "parent" along path
- $d(u, x) = d(u, v) + w(v, x)$
- shortest path from $u$ to $v$ has $j - 1$ hops

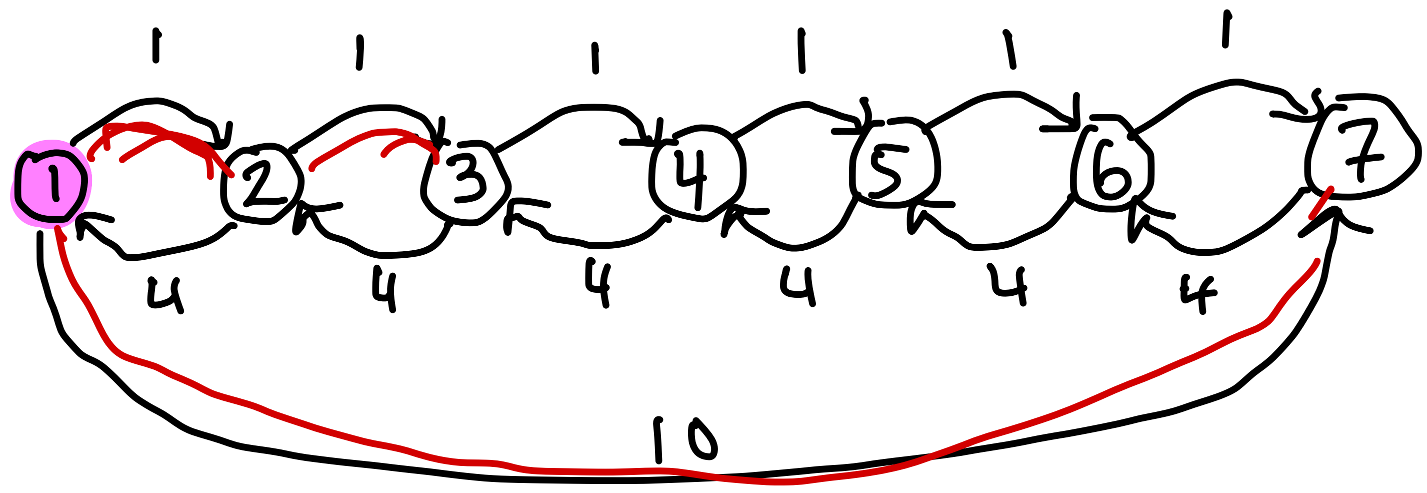
shortest path from $u$ to $x$



# Dynamic Programming Approach

**Idea.** For each vertex $v$ and each $j = 0, 1, 2, \ldots, n-1$ compute $\underline{d_j(u, v)}$ = length of shortest path from $u$ to $v$ with at most $j$ hops.

- Note $d(u, v) = d_{n-1}(u, v)$.



| $v$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $d_2(1,v)$ | 0 | 1 | 2 | $\infty$ | $\infty$ | 14 | 10 |

## Questions

**Question 1.** How to initialize $d_0(u, v)$?

$$d_0(u,v) = \begin{cases} 0 & u = v \\ \infty & \text{otherwise} \end{cases}$$

## Questions

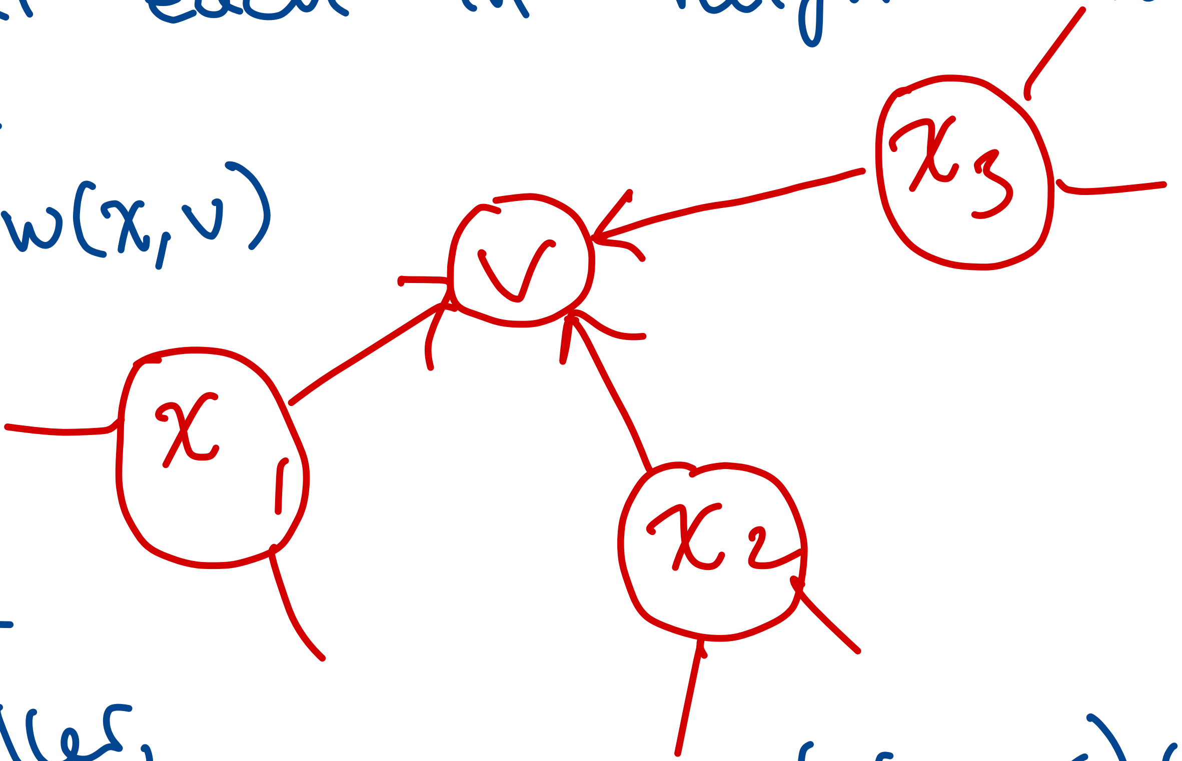
**Question 1.** How to initialize $d_0(u, v)$?

**Question 2.** Given $d_j(u, v)$ for all v, how to find $d_{j+1}(u, v)$?

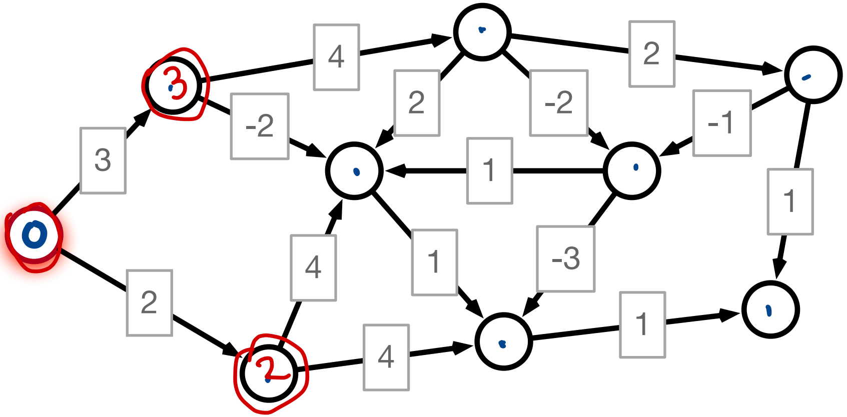
look at each "in" neighbor x

compare $d_j(u,x) + w(x,v)$ to cur est. of $d_{j+1}(u,v)$.

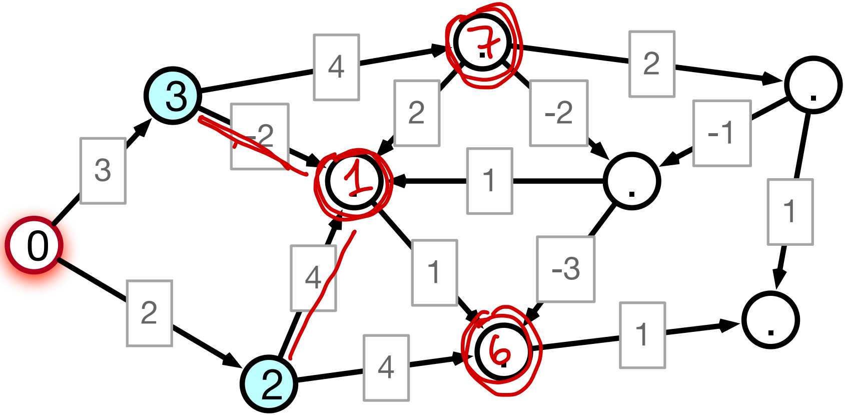
If smaller, update $d_{j+1}(u,v) \leftarrow d_j(u,x) + w(x,v)$.

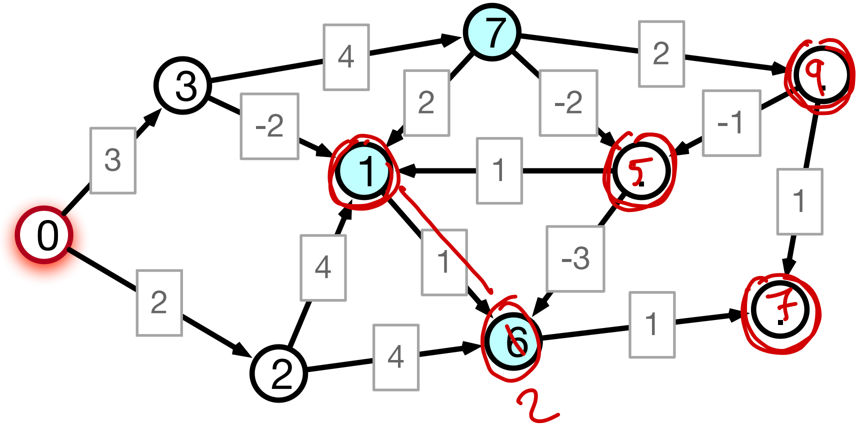


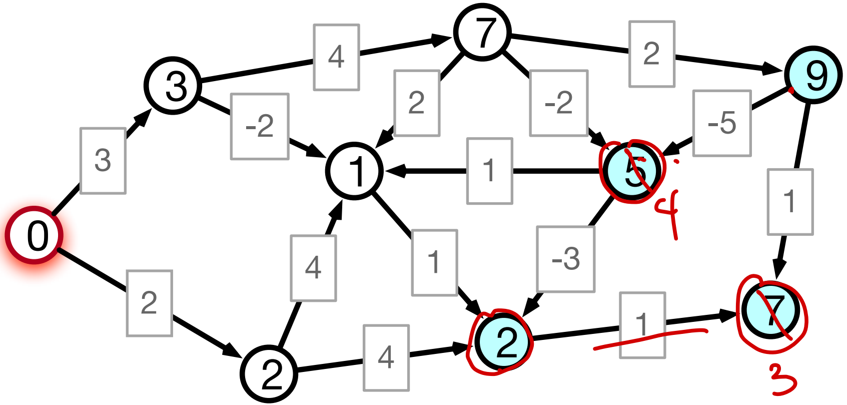
## Illustration

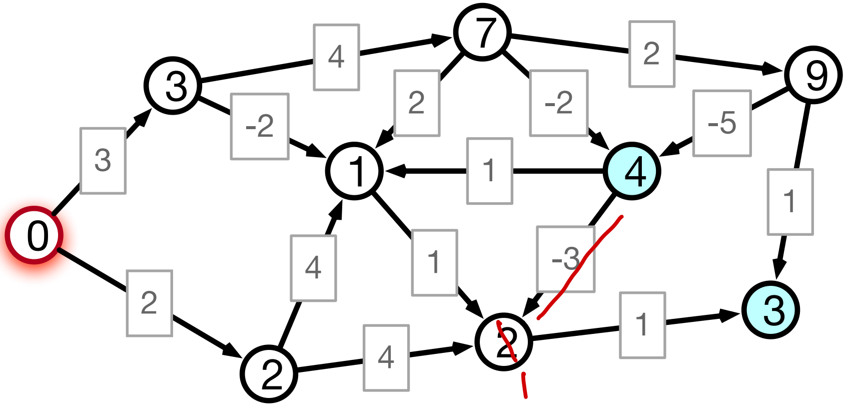


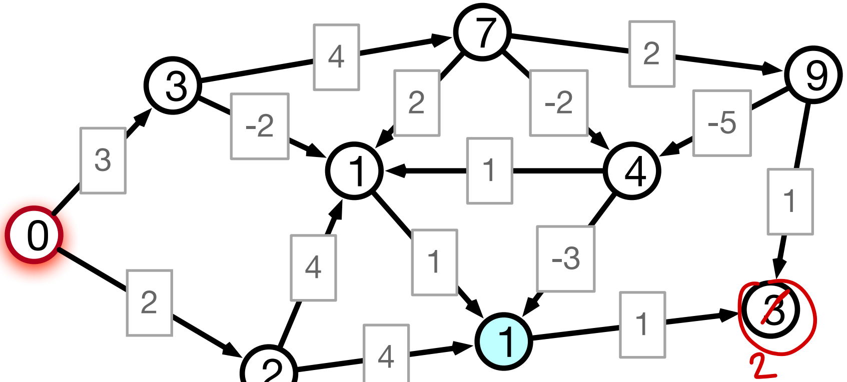
0
3
2
3
4
-2
2
-2
2
-1
1
1
1
4
4
-3
1
2
1

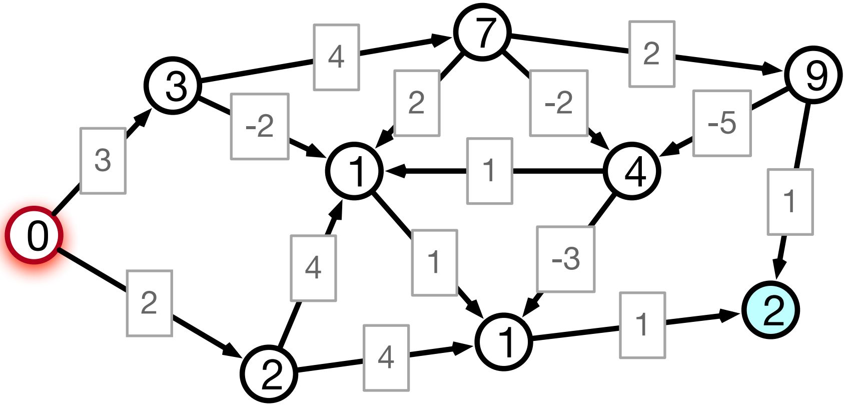

0
3
2
7
1
6
3
2
4
-2
2
-2
2
-1
1
4
1
-3
1
4
1
.
.
.
.
.
.

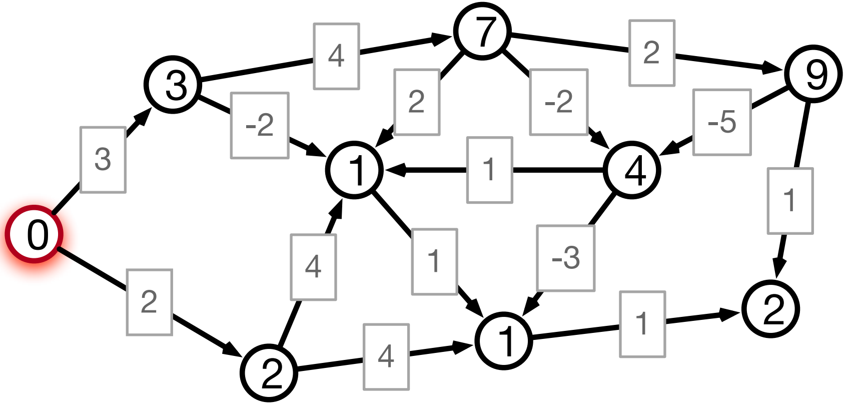

0
3
2
7
1
6
9
5
7
3
2
4
-2
4
4
2
-2
2
1
1
-1
-3
1
1
2


0
3
2
1
7
5
9
2
7
3
4
-2
2
-2
2
-5
1
1
4
1
-3
4
1
4
3


0
3
2
3
4
-2
1
4
2
7
2
-2
2
9
-5
4
1
1
-3
2
1
1
3
1


0
3
2
1
7
1
4
9
3
2
3
2
4
-2
4
4
2
1
1
-2
2
-3
-5
1
1


0
3
2
7
1
1
4
9
2
3
2
4
-2
4
4
2
-2
1
1
-3
2
-5
1
1


0
3
2
7
1
1
4
9
2
3
2
4
-2
4
4
2
-2
2
1
1
-3
-5
1
1

n vert
m edges

# Bellman-Ford Algorithm

# hops

vertex

$d[j,v] =$ j hop dist. from u to v

```
Bellman-Ford(V, E, w, u)
  d <- 2d array [0..n-1, 1..n]
  for v = 1 to n do d[0, v] <- infinity
  d[0, u] <- 0
  for j = 1 to n-1 do
    for each vertex v in V set d[j, v] <- d[j-1,v]
    for each vertex v in V
      for each neighbor x of v
        d[j, x] <- Min(d[j, x], d[j-1, v] + w[v, x])
  return d[n-1]
```

n-1

$O(m)$

Running time?

$O(n \cdot m)$

## Correctness

**Claim.** For all $j = 0, 1, \ldots, n-1$ and for all vertices $v$, $d[j, v]$ stores length of shortest path from $u$ to $v$ with $j$ or fewer hops. I.e., $d[j, v] = d_j(v)$

**Proof.** Induction on $j$.

*Base case*, $j = 0$.

## Inductive Step, $j - 1 \implies j$

- suppose $d[j-1, v] = d_{j-1}(v)$ for all $v$
- consider shortest path $P$ of $j$ hops from $u$ to $v$
- let $x$ be penultimate vertex in $P$
- then $d_j(v) = d_{j-1}(x) + w(x, v)$
- by inductive hypothesis, $d_{j-1}(x) = d[j-1, x]$
- therefore in iteration $j$, get $d[j, v] \leq d[j-1, x] + w(x, v) = d_{j-1}(x) + w(x, v) = d_j(v)$
- also have $d[j, v] \geq d_j(v)$ (why?)
- so $d[j, v] = d_j(v)$

## Conclusion

If $G$ has no negative weight cycles, then Bellman-Ford solves single source shortest paths in $O(mn)$ time.

# Dijkstra vs Bellman-Ford?

Running times:

- Dijkstra: $O(m \log n)$
- Bellman-Ford: $O(mn)$

Why pick Bellman-Ford over Dijkstra?

## Dijkstra vs Bellman-Ford?

Running times:

- Dijkstra: $O(m \log n)$
- Bellman-Ford: $O(mn)$

Why pick Bellman-Ford over Dijkstra?

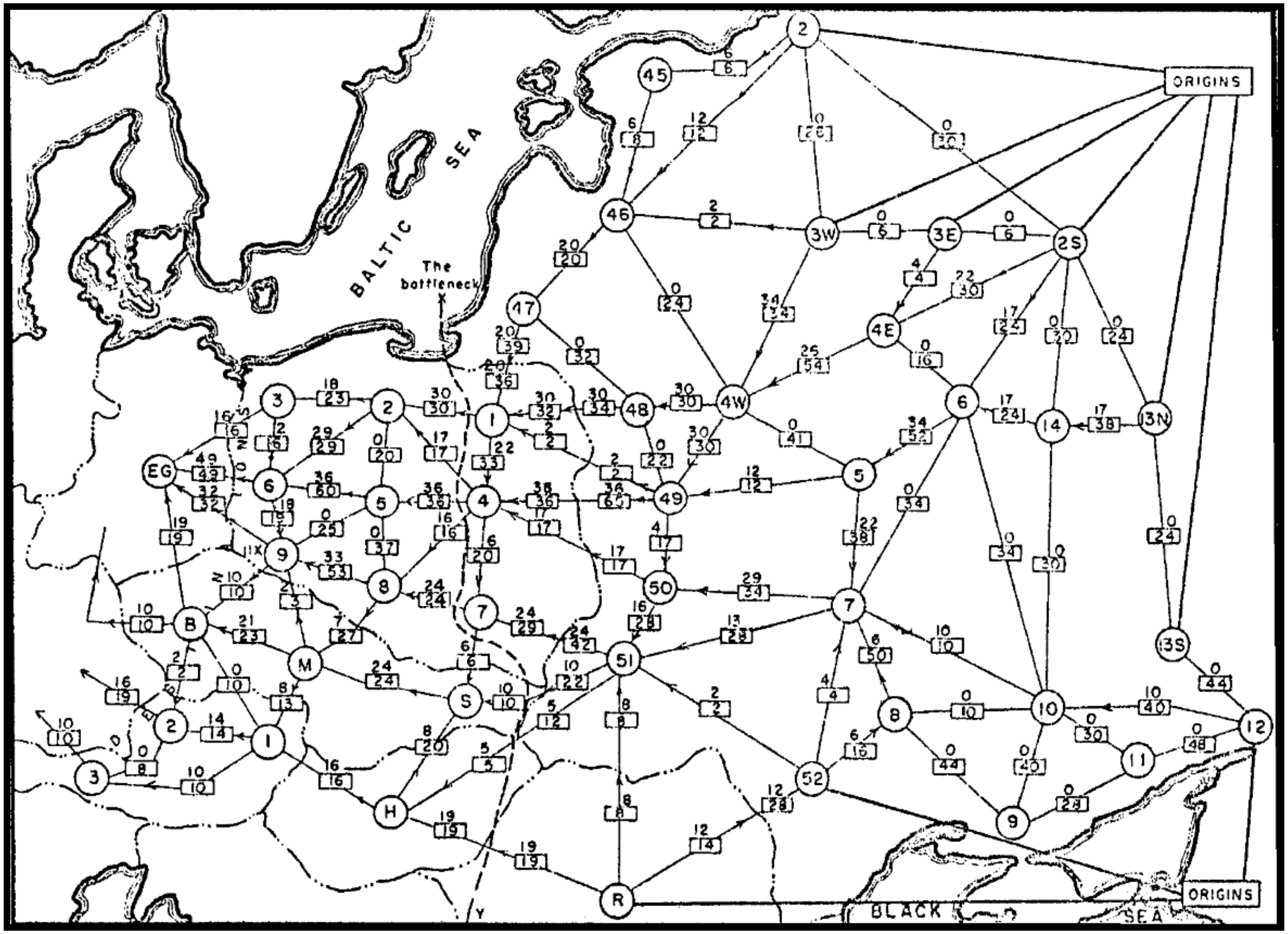
- Why might Bellman-Ford be preferable even if graph has no negative weight edges?

## Bellman-Ford Again

```
Bellman-Ford(V, E, w, u)
  d <- 2d array [0..n-1, 1..n]
  for v = 1 to n do d[0, v] <- infinity
  d[0, u] <- 0
  for j = 1 to n-1 do
    for each vertex v in V set d[j, v] <- d[j-1,v]
    for each vertex v in V
      for each neighbor x of v
        d[j, x] <- Min(d[j, x], d[j-1, v] + w[v, x])
  return d[n-1]
```

## Next Time: Cold War

# Rail Network of Eastern Europe



## Networks to Graphs

Modeling the network:

- nodes represent railway junctions
- edges represent rail lines
- weights represent capacities of lines
  - capacity indicates tonnage that can cross line per unit time
  - proportional to cost of disrupting movement along line

**Question 1.** How much material can the USSR transport to Western Europe per unit time?

**Question 2.** What is the cheapest way to disrupt flow of material?

## Network Flow

A new interpretation of directed graphs:

- network of (directional) pipes
- weights are *capacities*
    - how much fluid can flow through piper per time
- designated *source node s*
    - all edges directed away from *s*
- designated *sink* or *destination node t*
    - all edges directed towards *t*

**Question.** How much fluid be routed from *s* to *t* per unit time?