Lecture 16: Dijkstra's Algorithm

COSC 311 *Algorithms*, Fall 2022

Overview

1. Recap of ~~D~~BFS
2. Weighted Graphs
3. Weighted Shortest Paths
4. Dijkstra's Algorithm

vertices aka nodes

edges

Last Time

Unweighted Single-Source Shortest Paths:

- Given graph $G = (V, E)$ and starting vertex u
- Find for every vertex v, the distance $d(u, v)$
  - $d(u, v)$ = length of shortest path from u to v
  - shortest = fewest hops

Last Time

Unweighted Single-Source Shortest Paths:

- Given graph $G = (V, E)$ and starting vertex u
- Find for every vertex v, the distance $d(u, v)$
  - $d(u, v)$ = length of shortest path from u to v
  - shortest = fewest hops

Solution: Breadth-first Search (BFS)

- Process vertices in increasing order of distance from u

BFS Pseudocode

dist from u to v

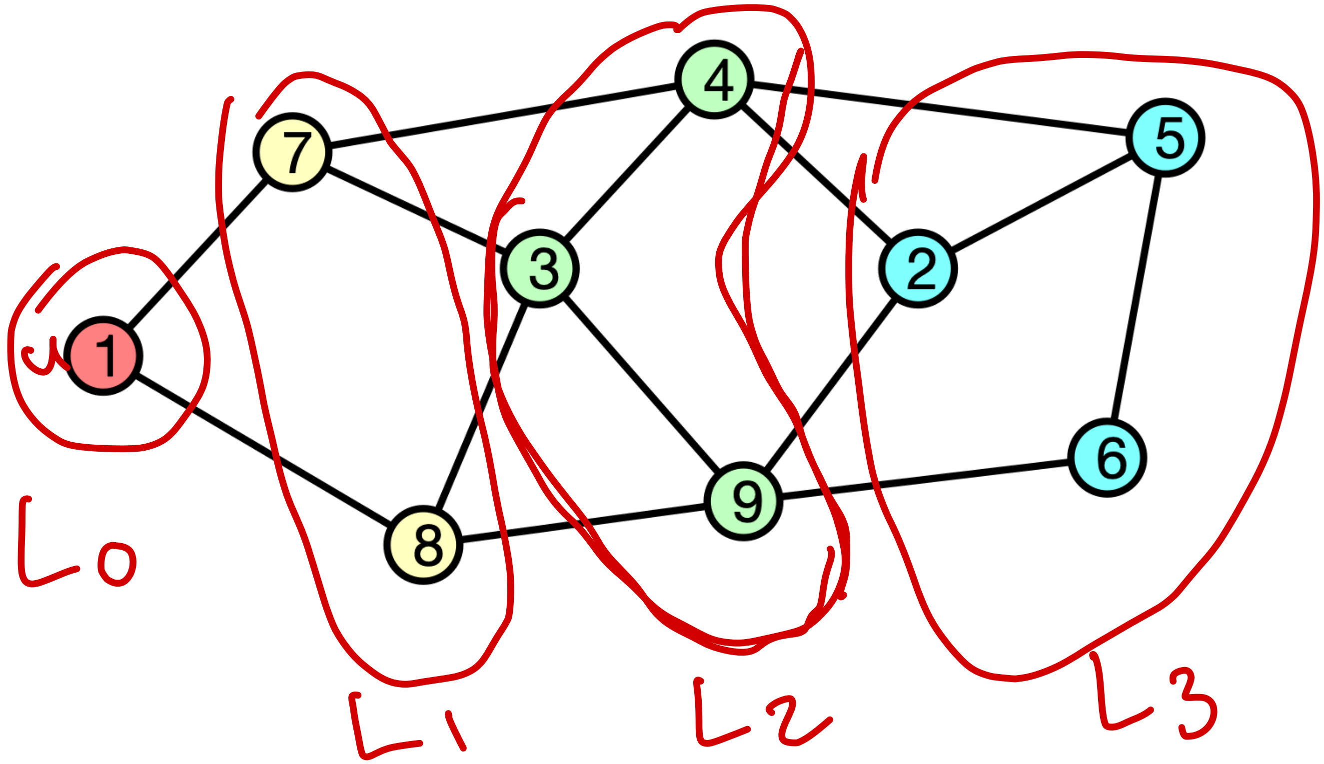
```
BFS(V, E, u):
  intialize d[v] <- -1 for all v
  d[u] <- 0
  queue.add(u)
  while queue is not empty do
    v <- queue.remove()
    for each neighbor w of v do
      if d[w] = -1 then
        d[w] <- d[v] + 1
        queue.add(w)
  return d
```

w has not been examined before

Correctness. Follows from interaction with queue: vertices added in order of increasing distance from u

- $\Longrightarrow$ distance is correct when vertex added

BFS Phases



queue	1	7	8	3	4	9	2	5	6

v	1	2	3	4	5	6	7	8	9
d[v]	0	3	2	2	3	3	1	1	2

n = # vertices, m = # edges in G

Ex: show sum of deg of G = $2m$

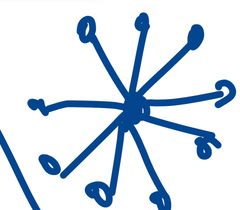
BFS Running Time?

```
BFS(V, E, u):
  intialize d[v] <- -1 for all v
 *d[u] <- 0
 *queue.add(u)
  while queue is not empty do
   * v <- queue.remove()
     for each neighbor w of v do
     *if d[w] = -1 then
     *  d[w] <- d[v] + 1
     *  queue.add(w)
  return d
```

$\Theta(n)$

* take $O(1)$ time

$\leq n-1$ inner iterations $\Rightarrow O(n^2)$ running



- 1 iteration per vertex
- for vertex v, inner loop iterates over neighbors $\Rightarrow$ $\deg(v)$ iterations

$$RT: \Theta(\underbrace{\deg(v_1) + \deg(v_2) + \ldots + \deg(v_n)}_{2m}) = \boxed{\Theta(m)}$$

$\Theta(m+n)$

More General Problem

Definition. A **weighted graph** is a graph $G(V, E)$ where each edge $e \in E$ is additionally assigned a (real valued) **weight** $w(e)$.

- for now, assume $w(e) \geq 0$

More General Problem

Definition. A **weighted graph** is a graph $G(V, E)$ where each edge $e \in E$ is additionally assigned a (real valued) **weight** $w(e)$.

- for now, assume $w(e) \geq 0$

Examples.

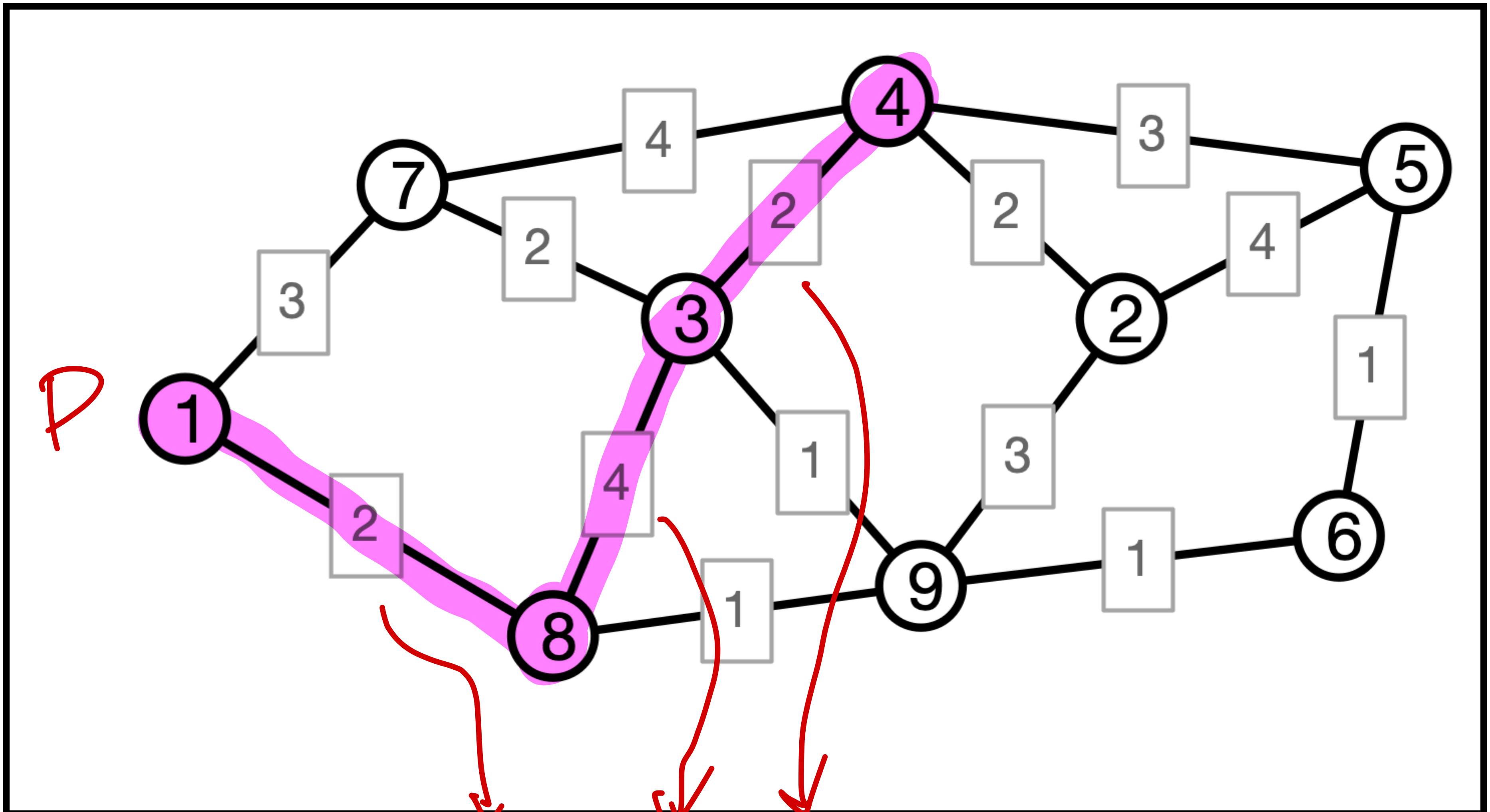
- weights = distances (not just number of hops)
- weights = cost of connection
- weights = latency of connection
- ...

Distance in Weighted Graphs

- $G = (V, E)$ a graph, w weights
- $P = v_0 e_1 v_1 e_2 v_2 \cdots e_k v_k$ a path
- The **(weighted) length** of P is

$$w(P) = w(e_1) + w(e_2) + \cdots + w(e_k)$$

Example

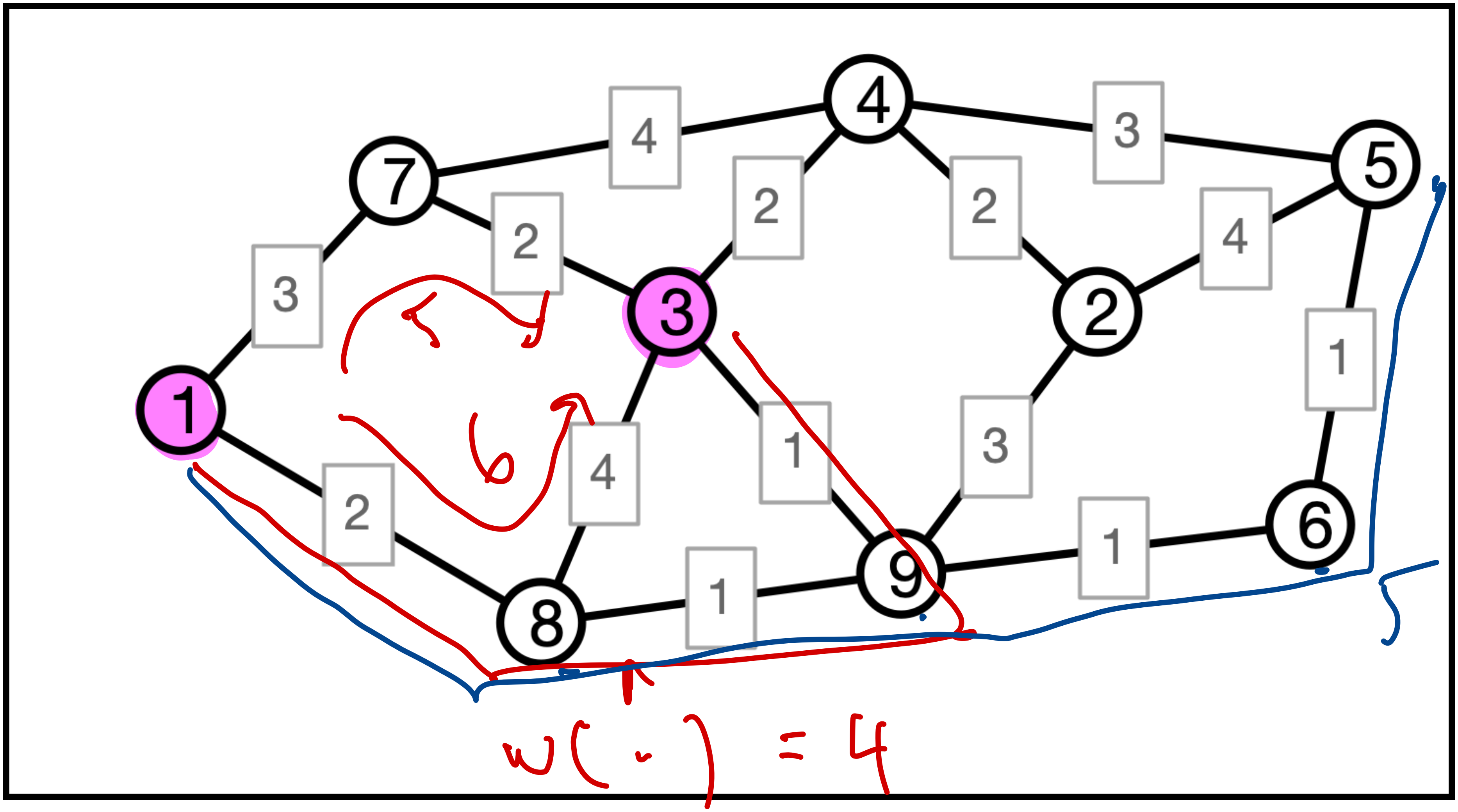


$w(P) = 2 + 4 + 2 = 8$

Weighted Shortest Paths

Given weights w, define $d_w(u, v)$ to be minimum (weighted) length of any path P from u to v.

Example



What is $d_w(1, 3)$? What about $d_w(1, 5)$?

Weighted SSSP = Single Source Shortest Path

Input.

- a weighted Graph $G = (V, E)$, edge weights w
- an initial vertex $u \in V$
- each vertex $v \in V$ has associated **adjacency list**
  - list of v's neighbors
  - includes weight of edge from v to each neighbor

Output.

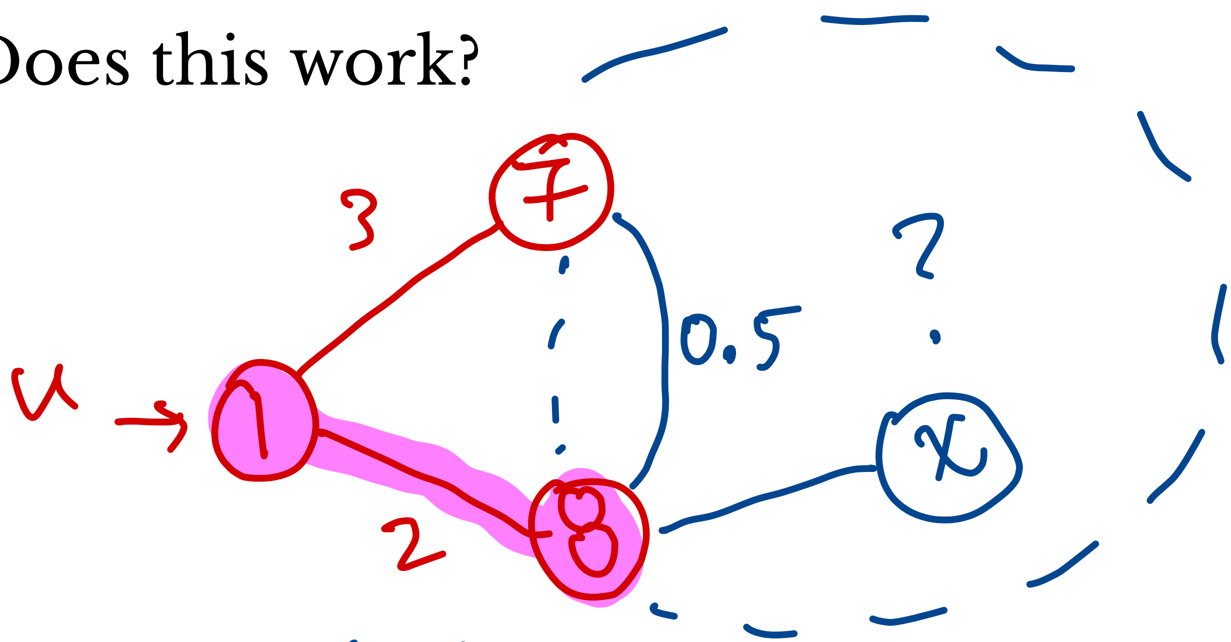
- A **map** $d : V \to \mathbf{R}$ such that $d[v] = d_w(u, v)$ is the graph distance from u to v
  - $d[v] = \infty$ indicates no path from u to v

Weighted SSSP

Does BFS compute *weighted* distances from u?

- must update procedure
- when processing edge (v, x), should update $d[x] \leftarrow d[v] + w(v, x)$ rather than setting $d[x] \leftarrow d[v] + 1$

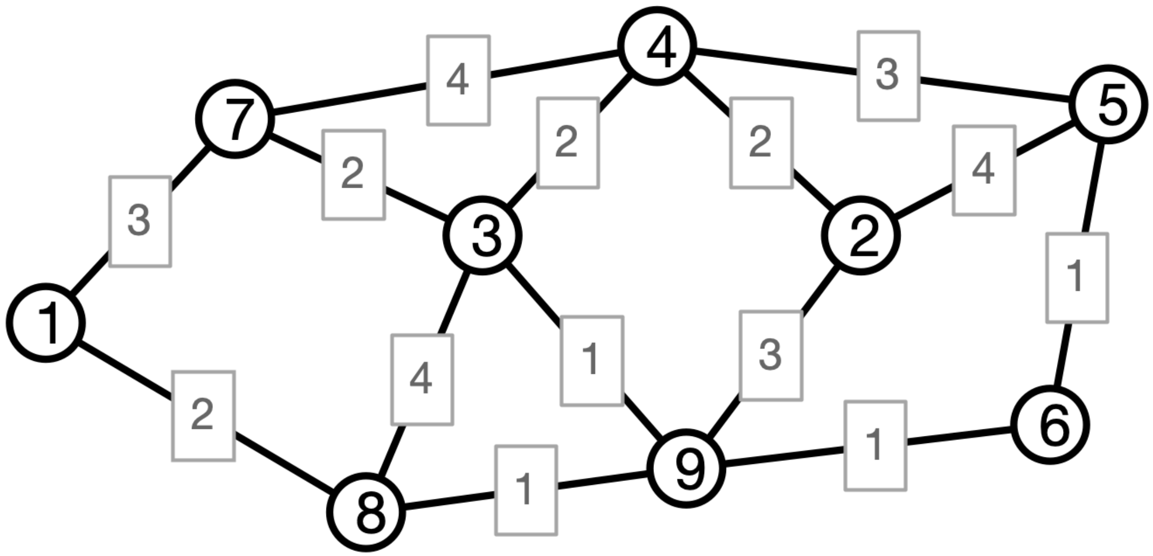
Does this work?



Must be shortest path?

Weighted BFS Example

Exercise: Check BFS gives wrong ans w/ weights.



cur	

queue	1

v	1	2	3	4	5	6	7	8	9
d[v]	0	-1	-1	-1	-1	-1	-1	-1	-1

Issue

- BFS processes vertices in order of fewest hops from u
- With weighted graphs, shortest path need not have fewest hops

BFS Analysis Takeaway

- BFS succeeds on unweighted graphs because closer vertices are processed before farther vertices
- Could we get similar behavior for weighted distances?

BFS Analysis Takeaway

- BFS succeeds on unweighted graphs because closer vertices are processed before farther vertices
- Could we get similar behavior for weighted distances?
  - must ensure: vertices processed in order of *weighted distance* from u
  - how can we do this?

BFS Analysis Takeaway

- BFS succeeds on unweighted graphs because closer vertices are processed before farther vertices
- Could we get similar behavior for weighted distances?
  - must ensure: vertices processed in order of *weighted distance* from u
  - how can we do this?
- How could we efficiently implement a modified procedure?

Dijkstra's Algorithm

Idea. Process elements in order of weighted distance from u

- Maintain set S of nodes whose distances from u is known
- Find element $\boxed{x \in V - S}$ that is closest to u and add it to S

x in V but not S

d[v]: dist from u to v

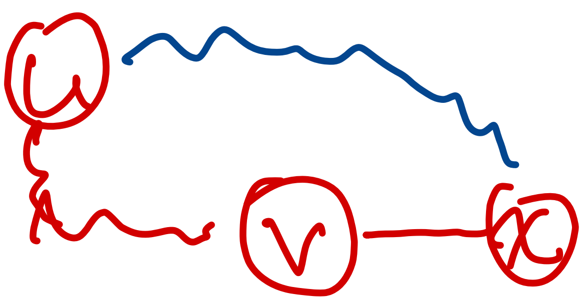
Dijkstra's Algorithm in Detail

1. Initialize $\boxed{d[u]} = 0$ and $d[v] = \infty$ for all $v \neq u$
2. Maintain set S of *finalized* nodes, initially empty
3. Process nodes. While $\boxed{S \neq V}$ do:
   - find node v in $V - S$ with minimal $d[v]$
   - add v to S
   - for each neighbor x of v
     - update $d[x] \leftarrow \min(d[x], d[v] + w(v, x))$

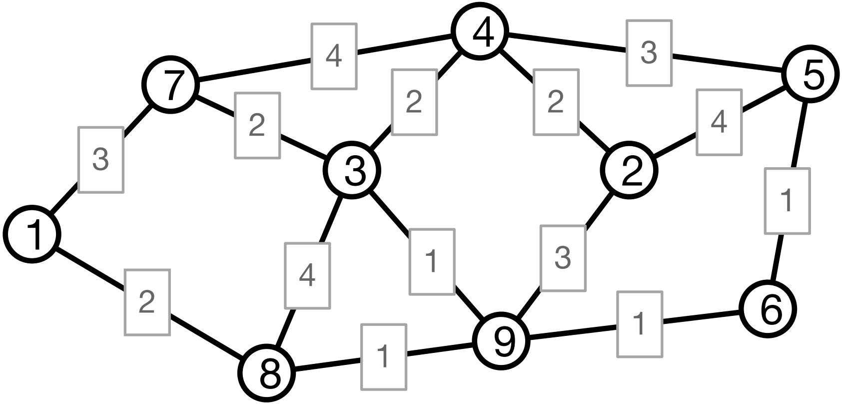
some nodes not finalized

prev estimate of distance

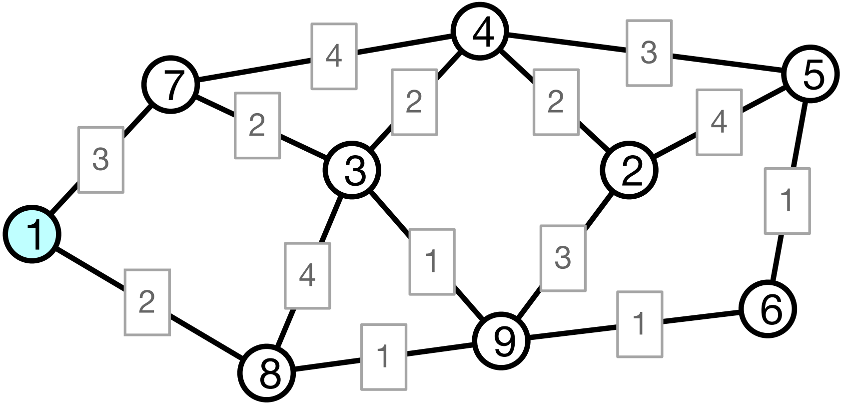
length of min path from u to x ending w/ edge (v,x)



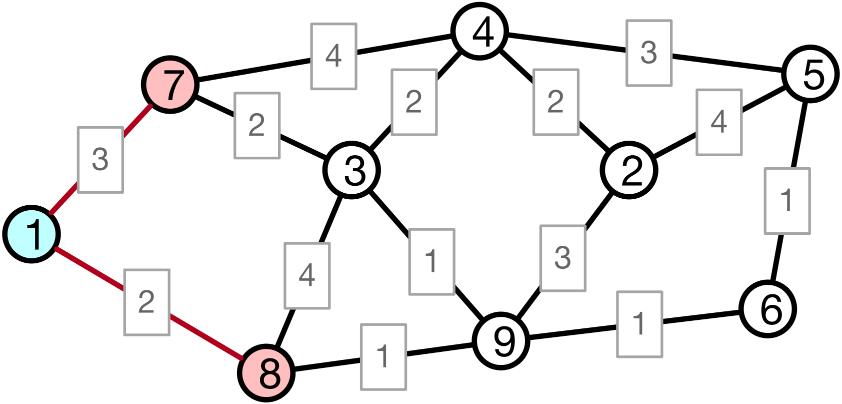
Dijkstra Illustration



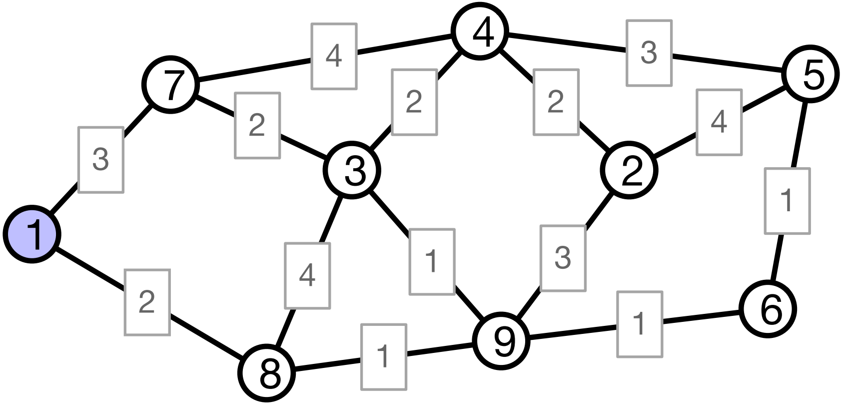
v	1	2	3	4	5	6	7	8	9
d[v]	0	.	.	.	.	.	.	.	.



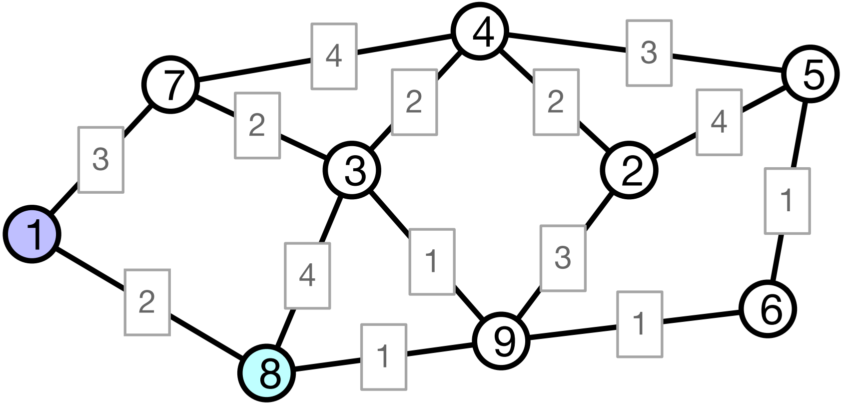
v	1	2	3	4	5	6	7	8	9
d[v]	0	.	.	.	.	.	.	.	.



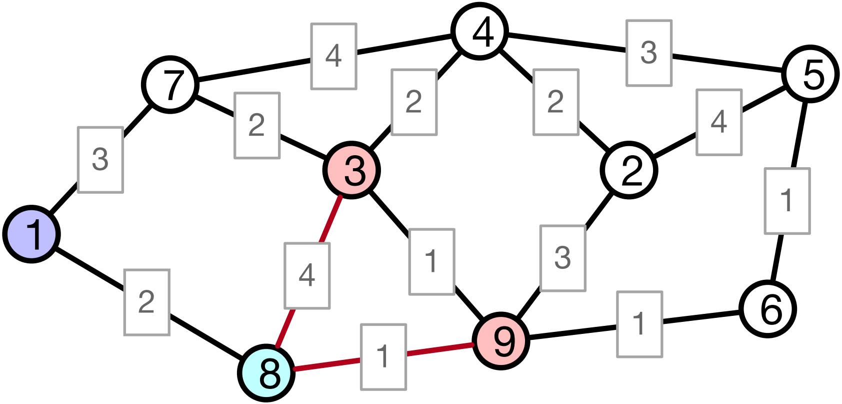
v	1	2	3	4	5	6	7	8	9
d[v]	0	.	.	.	.	.	3	2	.



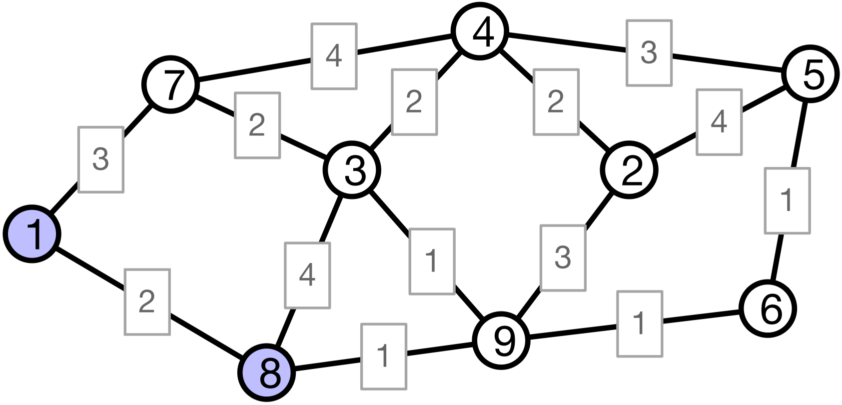
v	1	2	3	4	5	6	7	8	9
d[v]	0	.	.	.	.	.	3	2	.



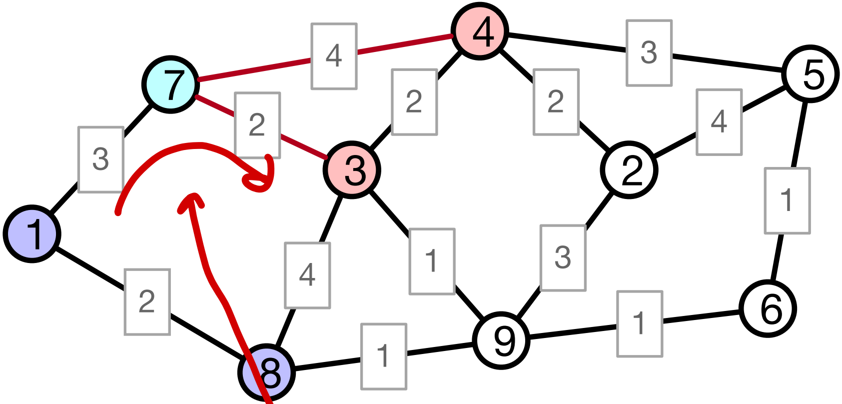
v	1	2	3	4	5	6	7	8	9
d[v]	0	.	.	.	.	.	3	2	.



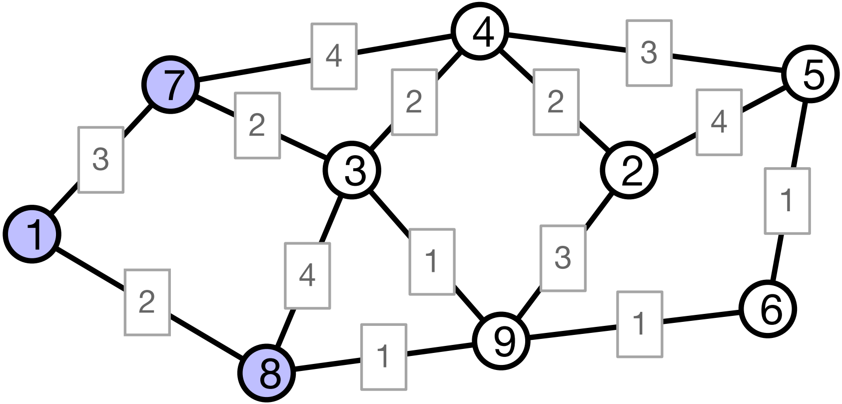
v	1	2	3	4	5	6	7	8	9
d[v]	0	.	6	.	.	.	3	2	3



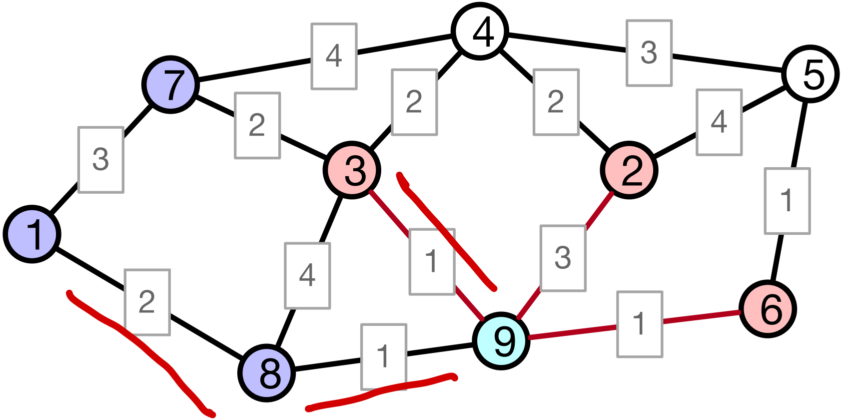
v	1	2	3	4	5	6	7	8	9
d[v]	0	.	6	.	.	.	3	2	3



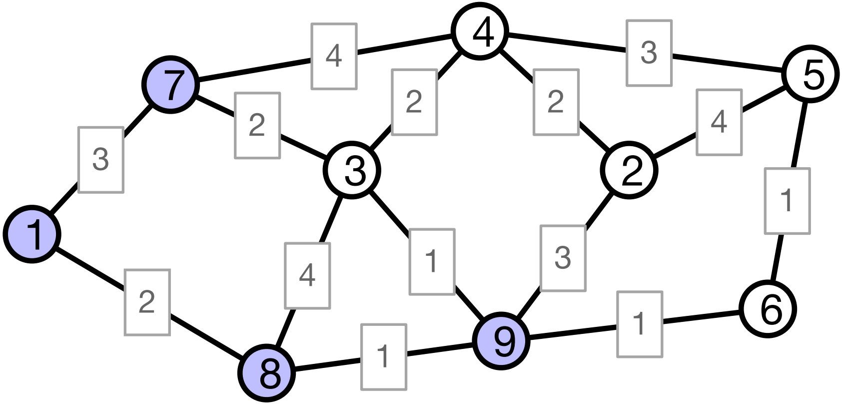
v	1	2	3	4	5	6	7	8	9
d[v]	0	.	5	7	.	.	3	2	3



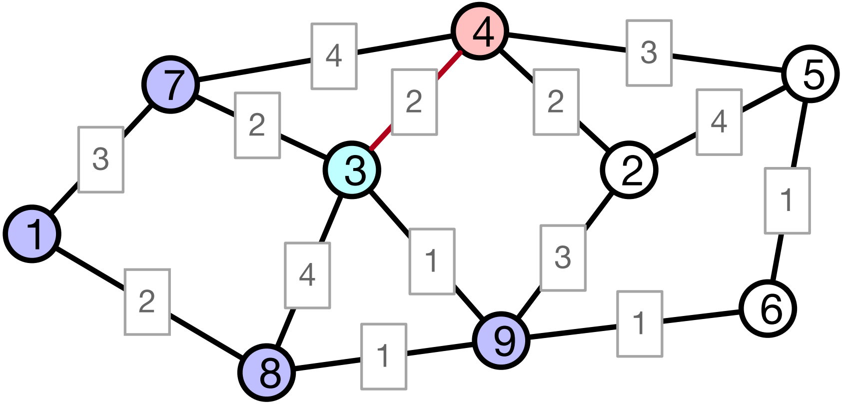
v	1	2	3	4	5	6	7	8	9
d[v]	0	.	5	7	.	.	3	2	3



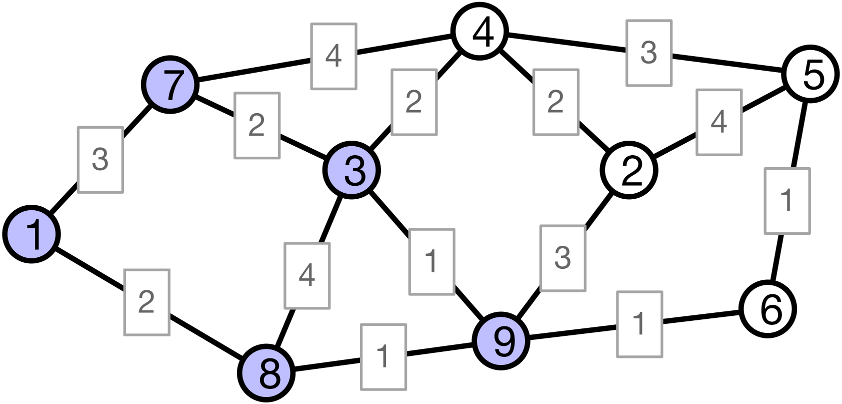
v	1	2	3	4	5	6	7	8	9
d[v]	0	6	4	7	.	4	3	2	3



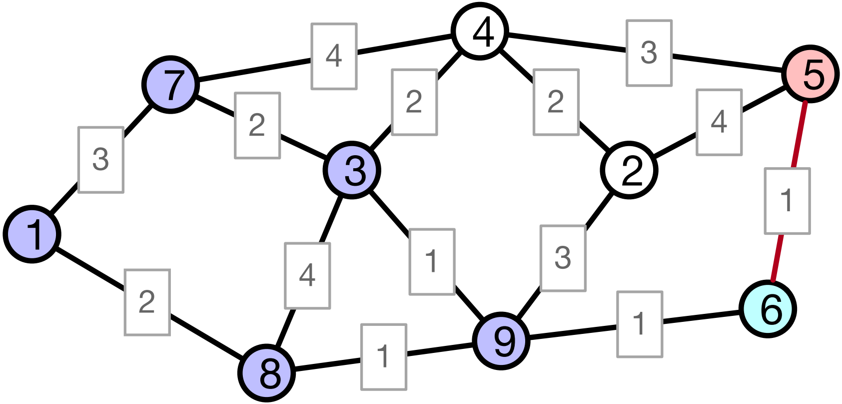
v	1	2	3	4	5	6	7	8	9
d[v]	0	6	4	7	.	4	3	2	3



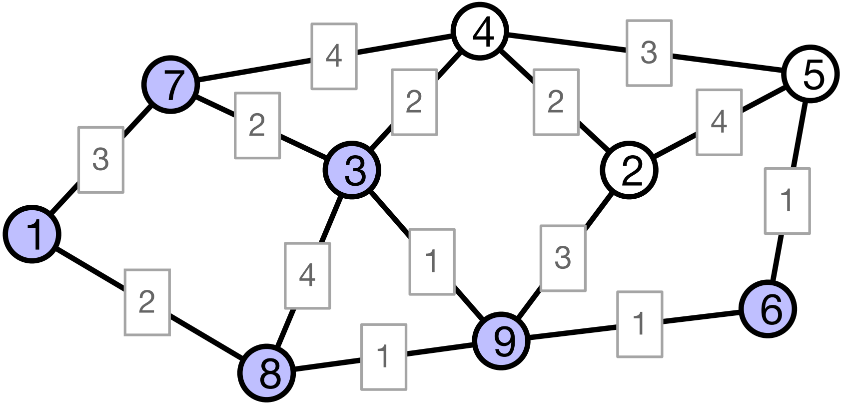
v	1	2	3	4	5	6	7	8	9
d[v]	0	6	4	6	.	4	3	2	3



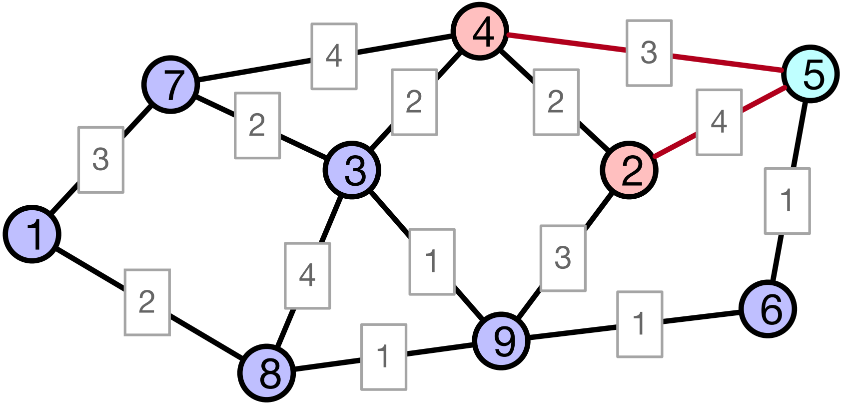
v	1	2	3	4	5	6	7	8	9
d[v]	0	6	4	6	.	4	3	2	3



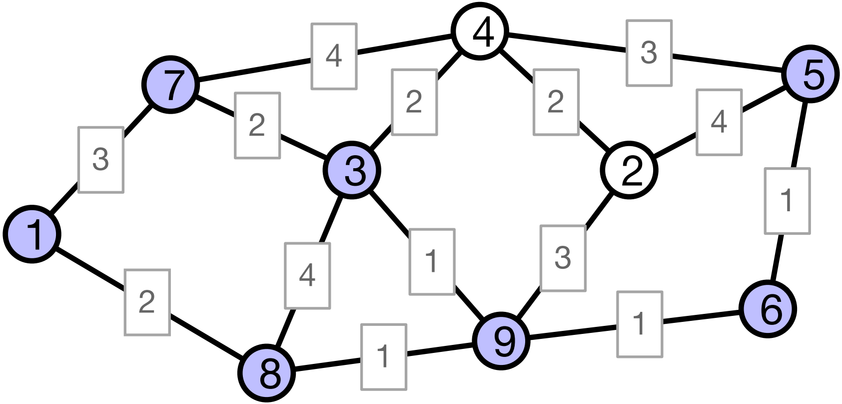
v	1	2	3	4	5	6	7	8	9
d[v]	0	6	4	6	5	4	3	2	3



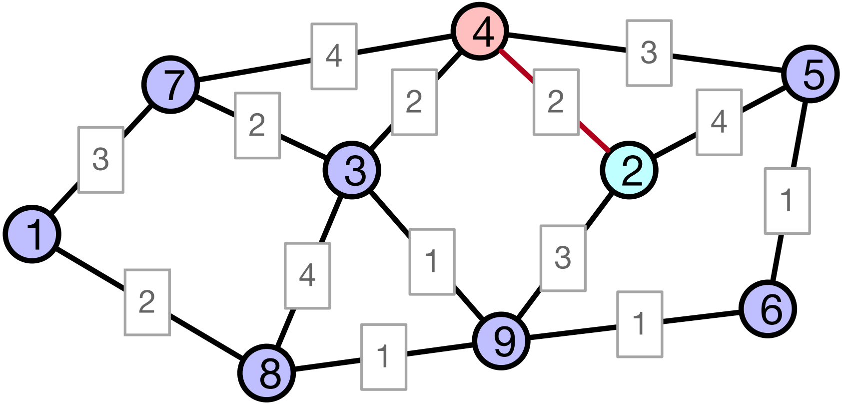
v	1	2	3	4	5	6	7	8	9
d[v]	0	6	4	6	5	4	3	2	3



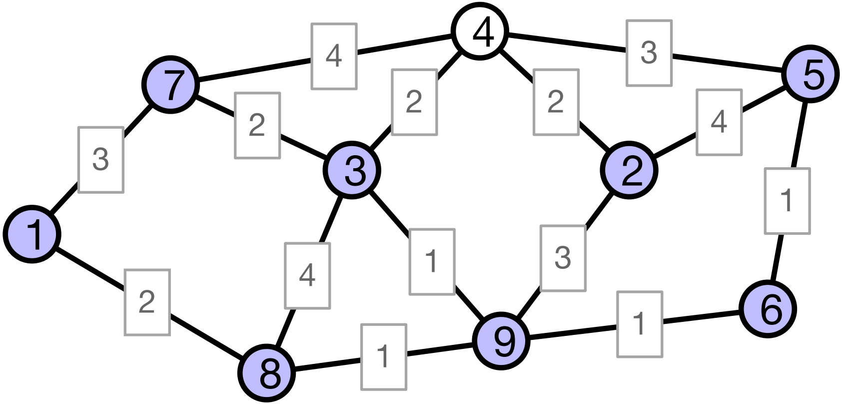
v	1	2	3	4	5	6	7	8	9
d[v]	0	6	4	6	5	4	3	2	3



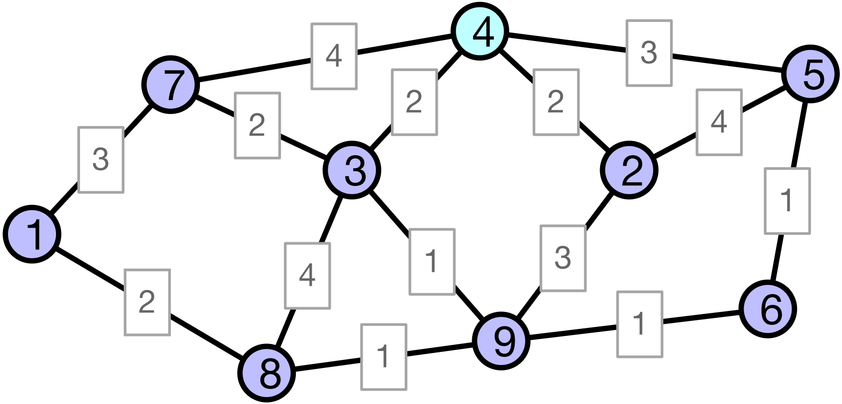
v	1	2	3	4	5	6	7	8	9
d[v]	0	6	4	6	5	4	3	2	3



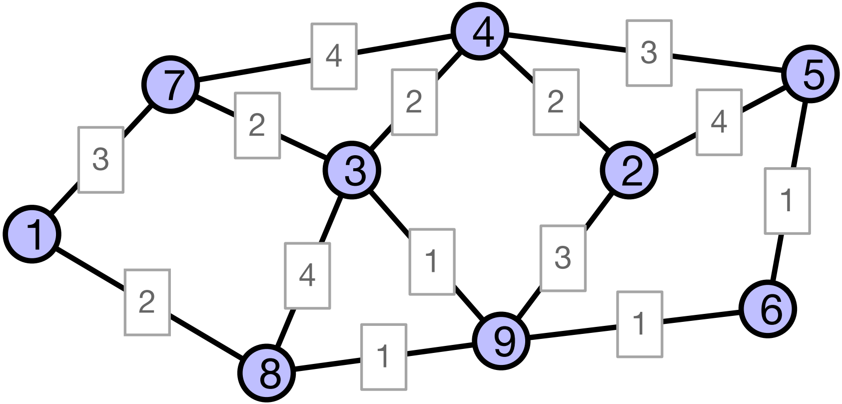
v	1	2	3	4	5	6	7	8	9
d[v]	0	6	4	6	5	4	3	2	3



v	1	2	3	4	5	6	7	8	9
d[v]	0	6	4	6	5	4	3	2	3



v	1	2	3	4	5	6	7	8	9
d[v]	0	6	4	6	5	4	3	2	3



Claim: these are correct distances from 1.

v	1	2	3	4	5	6	7	8	9
d[v]	0	6	4	6	5	4	3	2	3

Correctness

1. Initialize $d[u] \leftarrow 0$ and $d[v] \leftarrow \infty$ for all $v \neq u$
2. Maintain set S of *finalized* nodes, initially empty
3. Process nodes: while $S \neq V$
   - find node v in $V - S$ with minimal $d[v]$
   - add v to S
   - for each neighbor x of v
     - update $d[x] \leftarrow \min(d[x], d[v] + w(v, x))$

Claim. For every vertex $v \in S$, $d[v]$ stores the correct (weighted) distance $d_w(u, v)$.

Proof of Claim

Claim. For every vertex $v \in S$, $d[v]$ stores the correct (weighted) distance $d_w(u, v)$.

Proof. Use induction on size of S. Set $k =$ size of S.

Base case $k = 1$. Only u is added to S. Set $d[u] \leftarrow 0$, which is correct answer.

Inductive Step I

Inductive hypothesis. When S contains k elements, $d[v]$ is correct for all vertices $v \in S$.

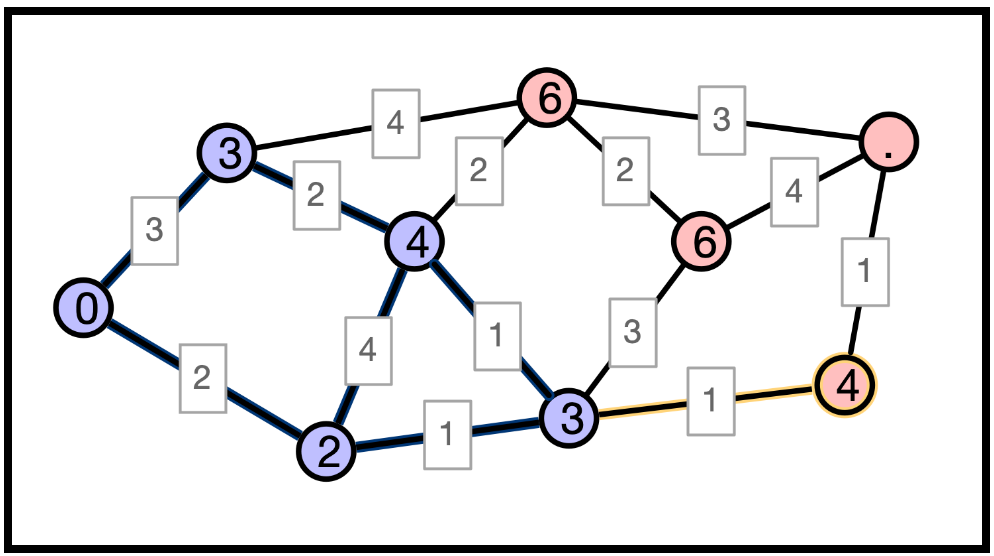
Consider next iteration of outer loop:

- x has $d[x] = \min_{v \in S}(d[v] + w(v, x))$

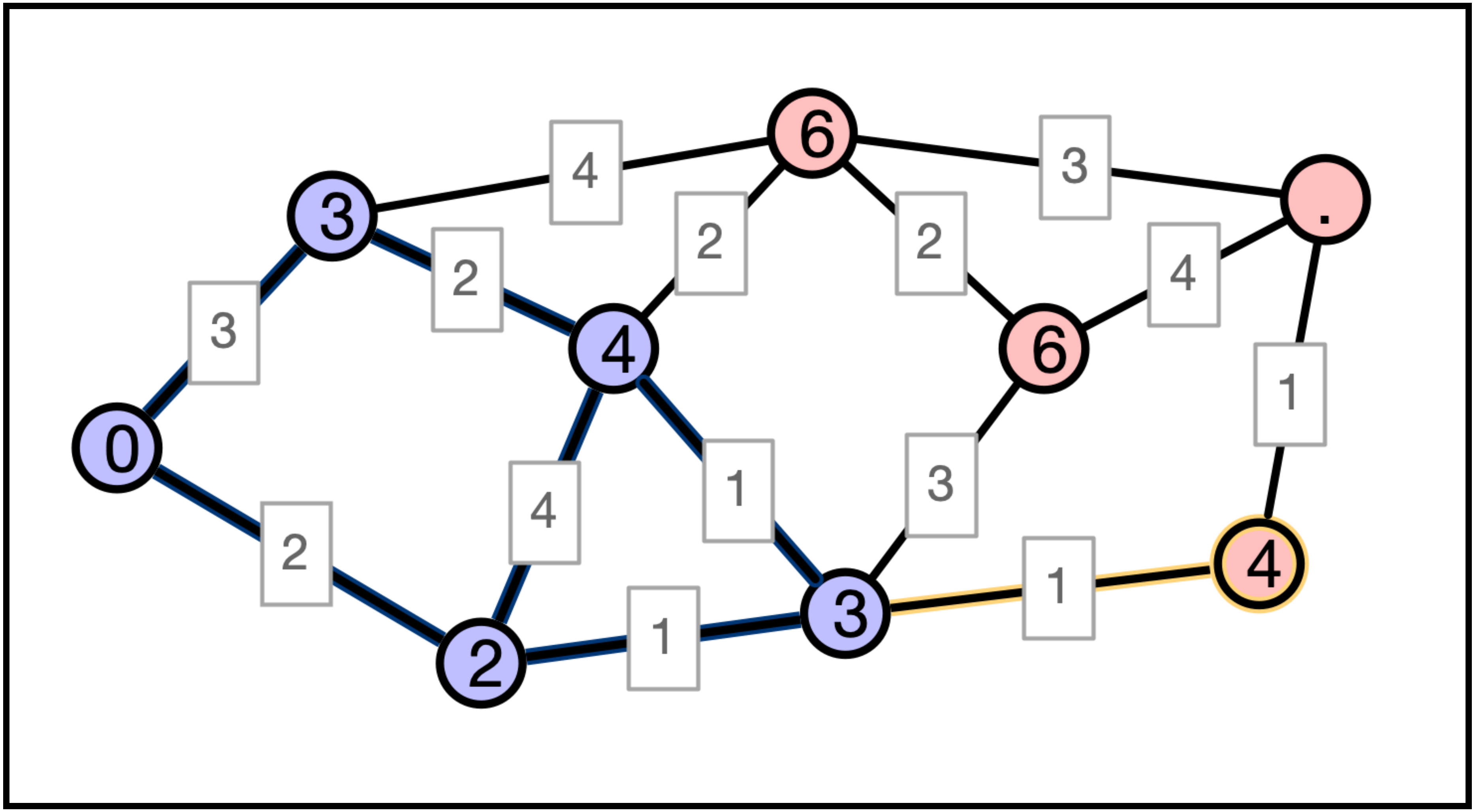
Inductive Step II

Must show: $d[x] = d_w(u, x)$; argue by *contradiction*

1. suppose $d[x] \neq d_w(u, x)$
2. observe: there is a path from u to x of length $d[x]$
3. $\implies d_w(u, x) < d[x]$
4. $\implies$ there is a path P from u to x of length $\ell < d[x]$



Shorter Path Illustration



Inductive Step III

Must show: $d[x] = d_w(u,x)$; argue by *contradiction*

1. suppose $d[x] \neq d_w(u,x)$
2. observe: there is a path from u to x of length $d[x]$
3. $\implies$ $d_w(u,x) < d[x]$
4. $\implies$ there is a path P from u to x of length $\ell < d[x]$
5. P must leave S at some point y
6. by definition of x, any path from u to y must be longer than $d[x]$
7. $\implies$ $w(P) \geq d[x]$, which contradicts 4

Conclusion. $d[x] = d_w(u,x)$, as claimed.

Next Time

1. Implementing Dijkstra's algorithm
    - how do we find x with minimum $d[x]$ *efficiently?*
    - review heaps/priority queues
2. Minimum spanning tree problem