Lecture 15: Graphs and Distances

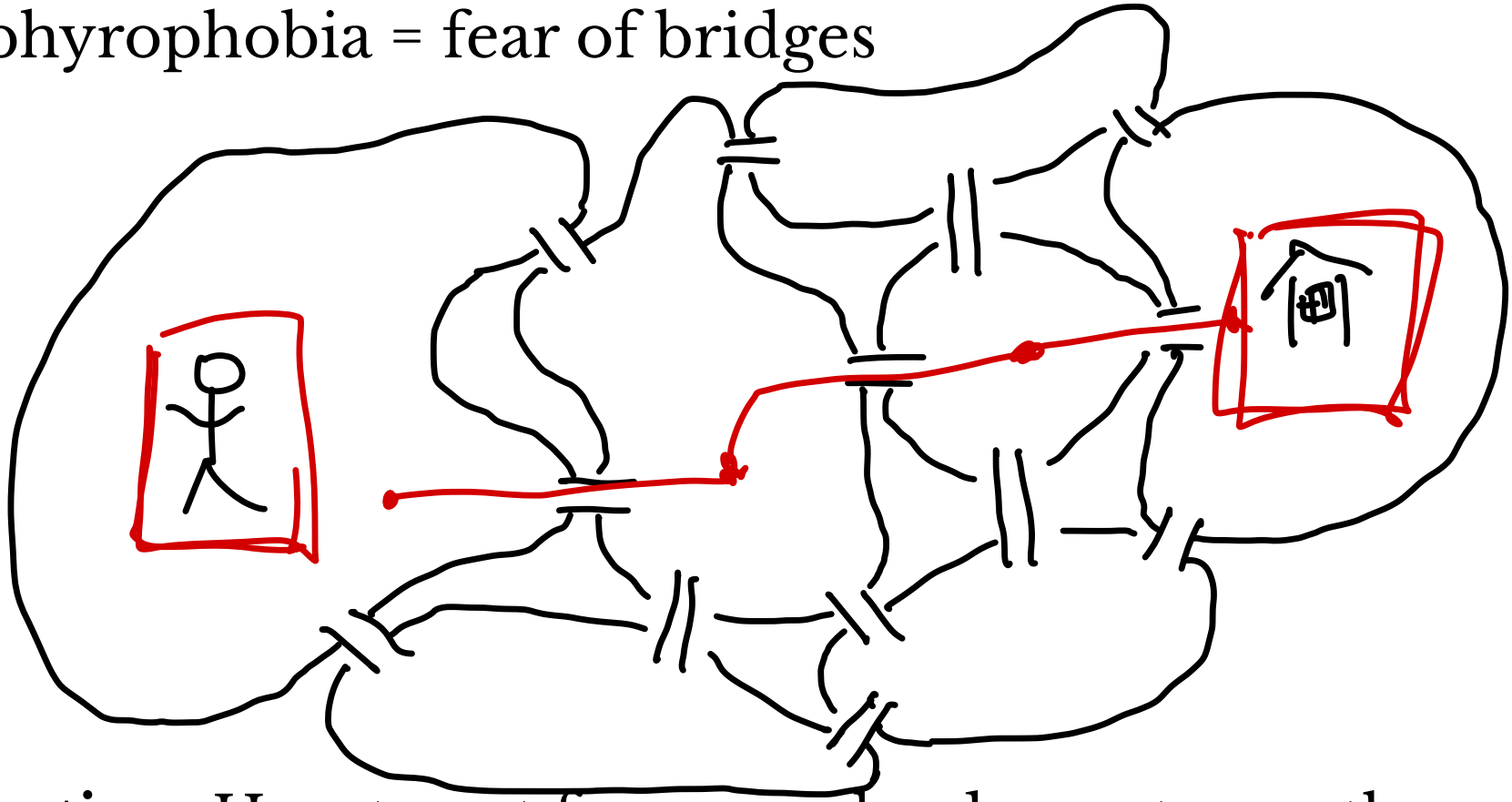
COSC 311 *Algorithms,* Fall 2022

Overview

1. Single Source Shortest Paths
2. Depth First Search
3. Weighted Graphs
4. Weighted Shortest Paths

More Bridges

Gephyrophobia = fear of bridges



Question. How to get from one landmass to another, crossing the fewest possible number of bridges?

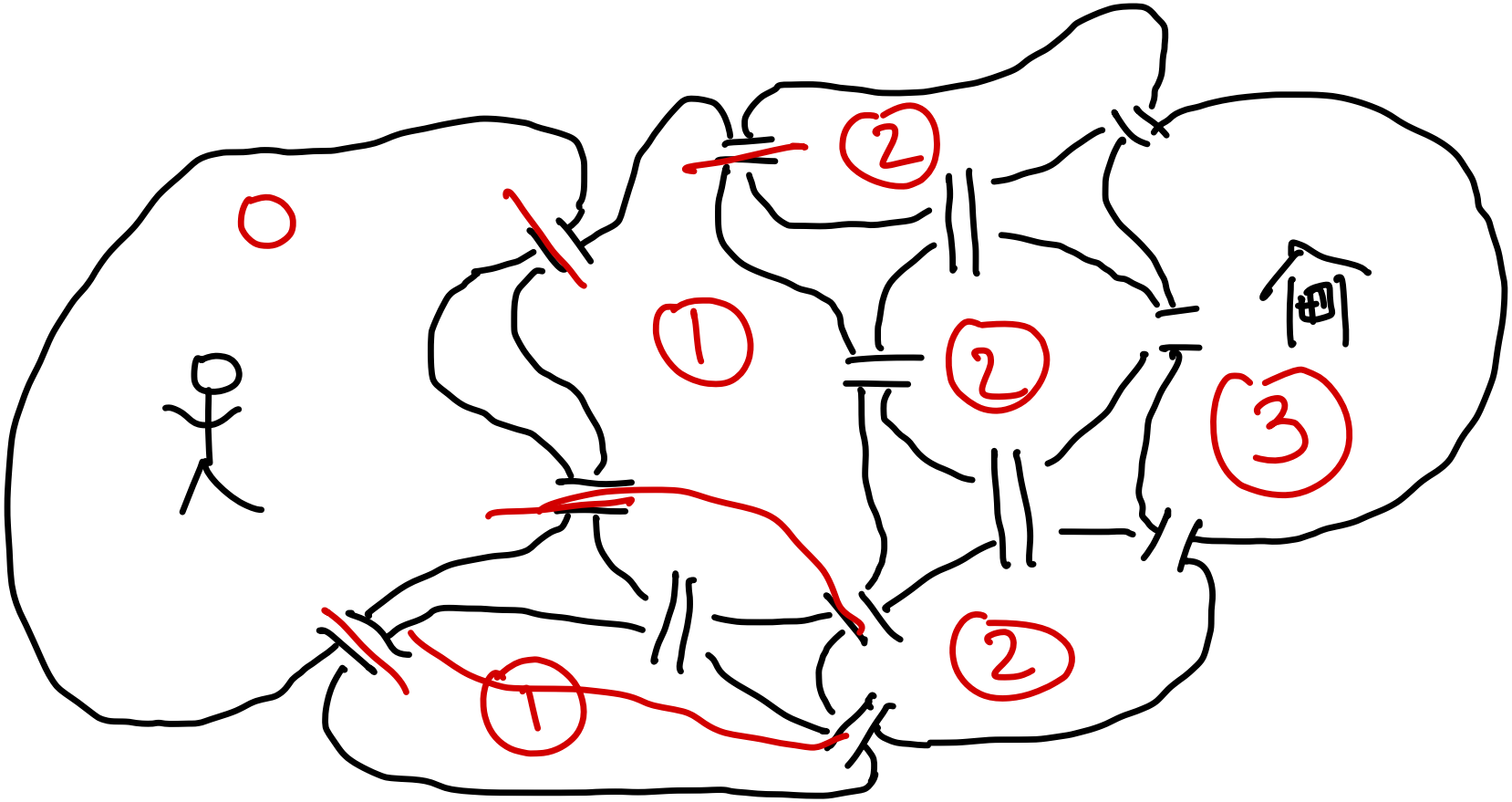
Strategy

Find shortest (fewest hops) route by:

1. find all vertices reachable in 1 hop
2. find all vertices reachable in 2 hops
3. find all vertices reachable in 3 hops
4. …

Continue until destination is found

Illustration



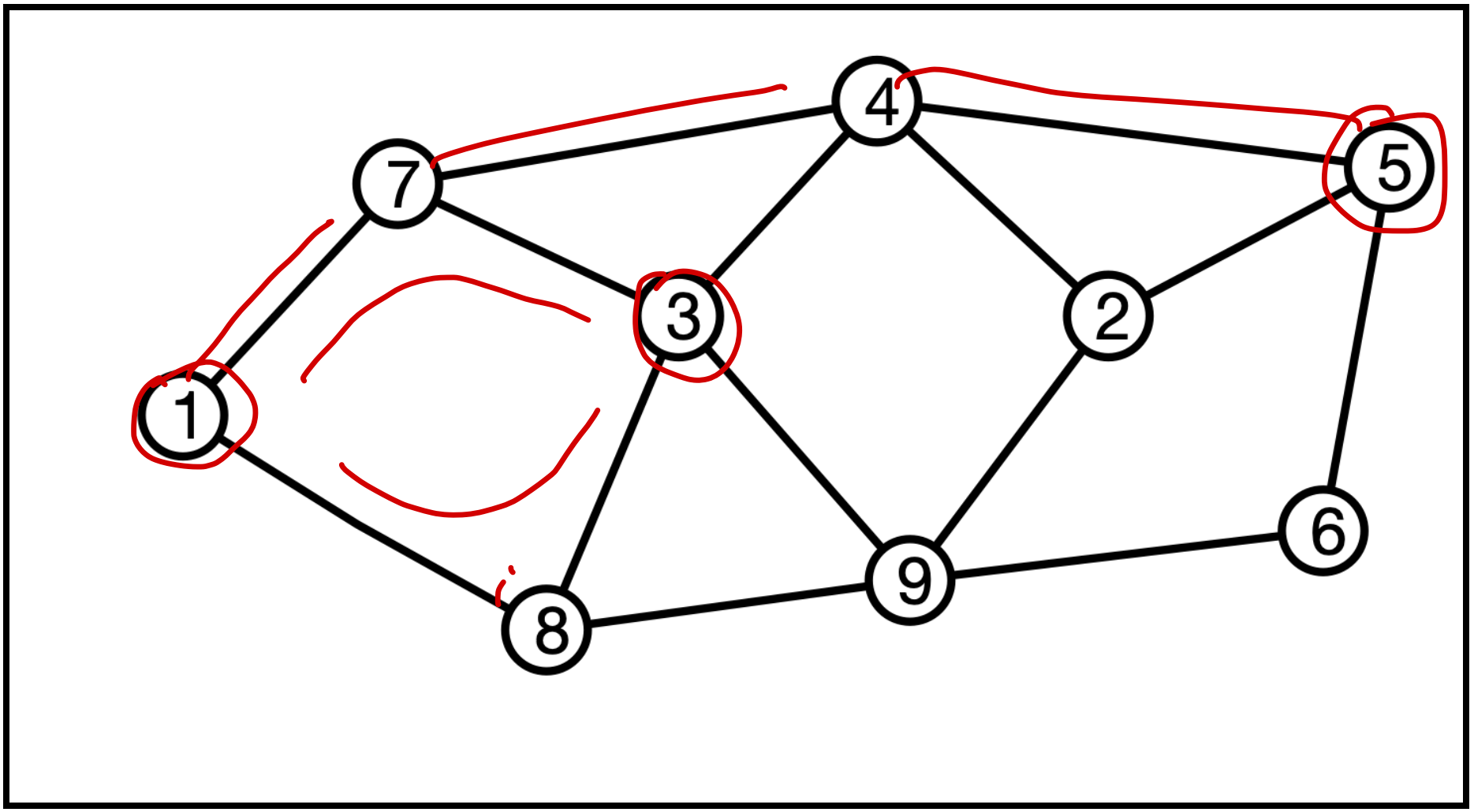
Graph Distances

vertices

edges

Definition. $G = (V, E)$ a graph, $u, v \in V$ vertices. The **graph distance** between u and v, denoted $d(u, v)$, is the length of the shortest path from u to v in G.

Example



What is $d(1,3)$? What is $d(1,5)$?

2

3

Single Source Shortest Paths (SSSP)

Unweighted version

Input.

- a Graph $G = (V, E)$
- an initial vertex $u \in V$
- each vertex $v \in V$ has associated **adjacency list**
  - list of v's neighbors

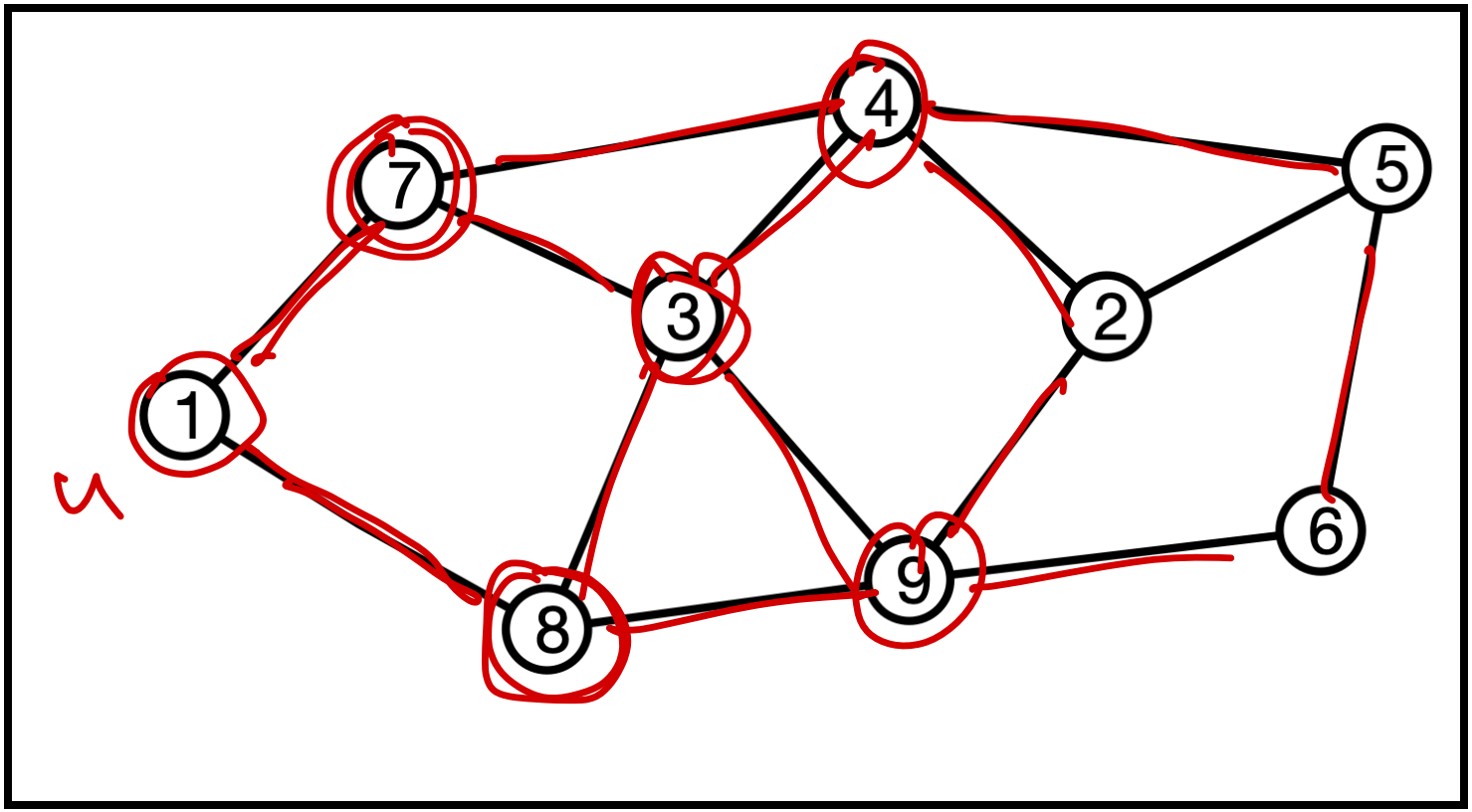
Output.

- A **map** $d_u : V \to \{-1, 0, 1, 2, \ldots\}$ such that $d_u(v) = d(u, v)$ is the graph distance from u to v
  - $d[v] = -1$ indicates no path from u to v

vertex # d[v] = v's dist. from u

sentinel value

Example



vert	1	2	3	4	5	6	7	8	9
dist	0	3	2	2	3	3	1	1	2

BFS Solution

Breadth-First Search

1. start at u
2. examine u's neighbors, at distance 1
3. examine u's neighbors' neighbors, at distance 2
4. ⋮

Greedily examine closest vertices that have not yet been examined...

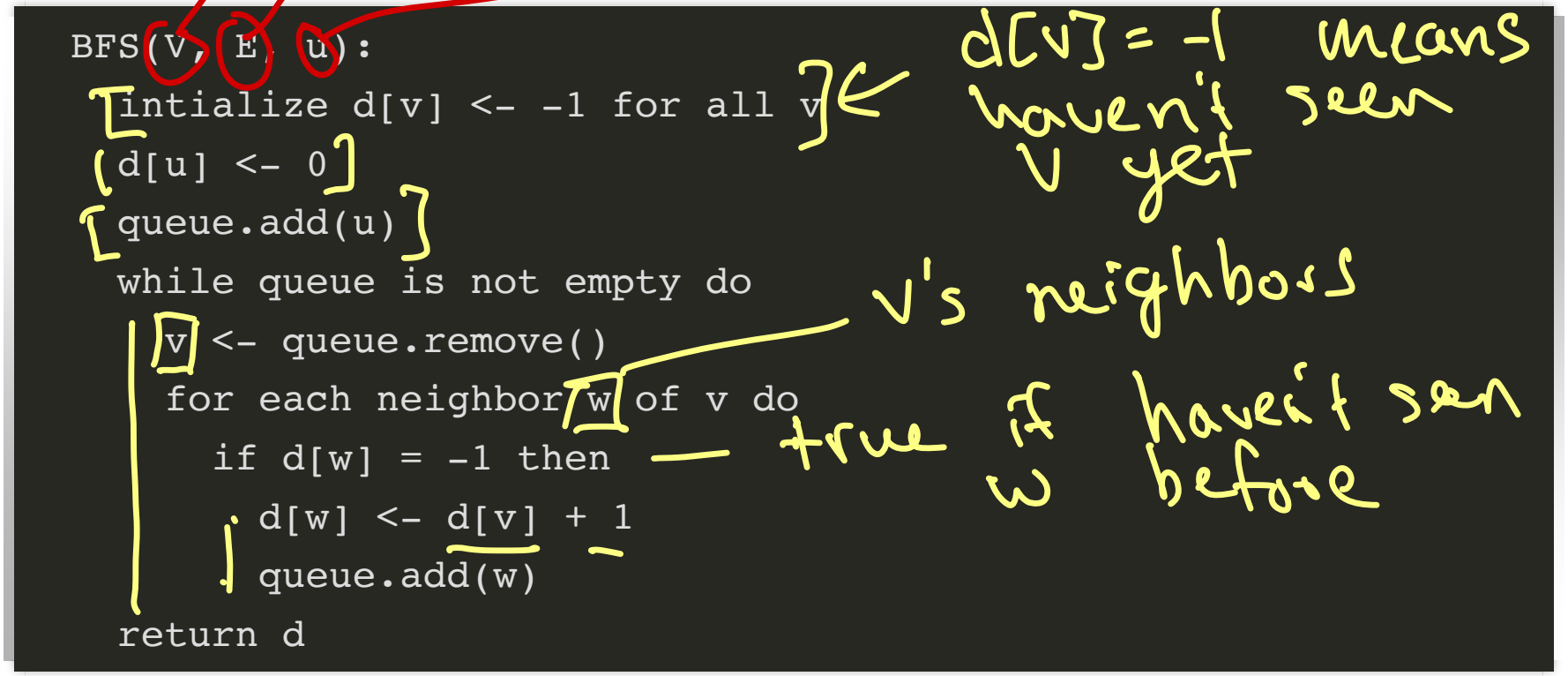
Queues

Abstract data type (ADT)

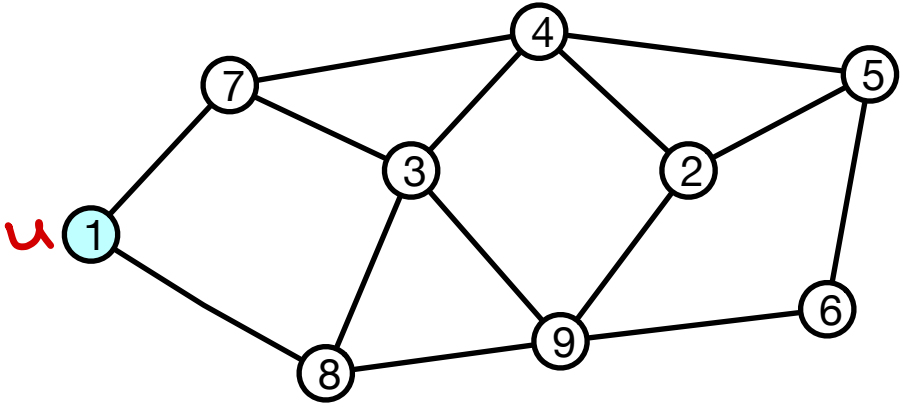
- stores elements
- two basic operations
  - `add(x)` adds element x to queue
  - `remove()` removes and returns element
- FIFO: first in, first out

order of removal = order of addition

BFS Pseudocode



BFS Illustration

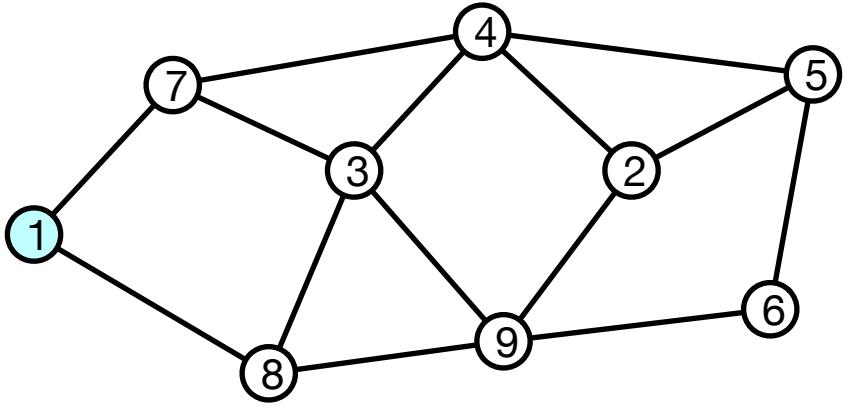


v

cur

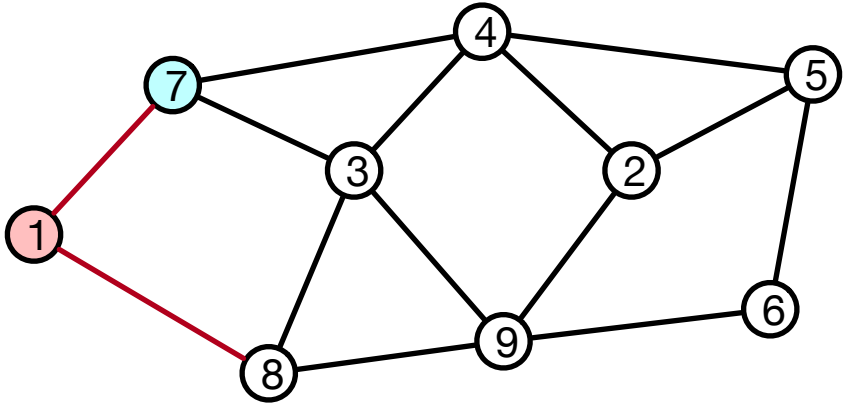
queue	1

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



cur	1
queue	1

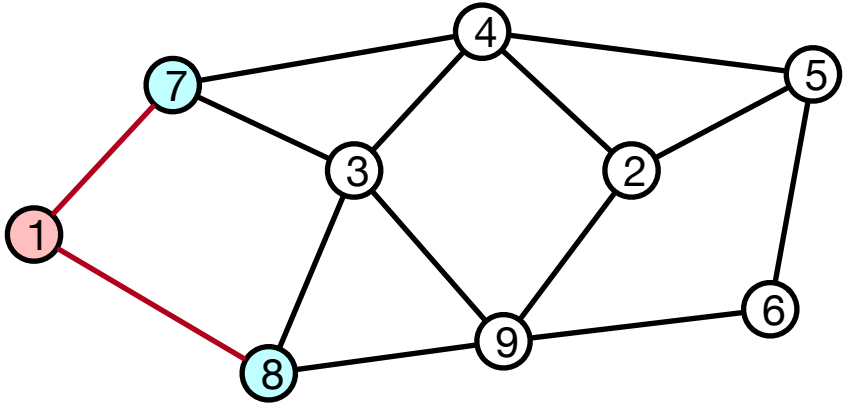
v	1	2	3	4	5	6	7	8	9
d[v]	0	-1	-1	-1	-1	-1	-1	-1	-1



cur	1

queue	7

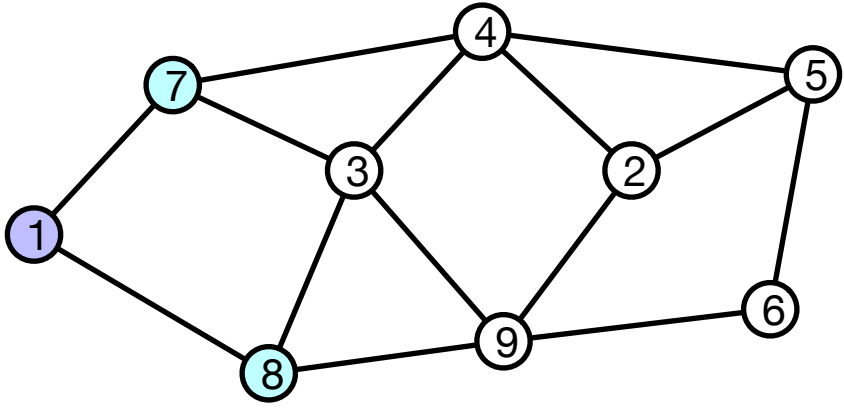
v	1	2	3	4	5	6	7	8	9
d[v]	0	-1	-1	-1	-1	-1	1	-1	-1



cur	1

queue	7	8

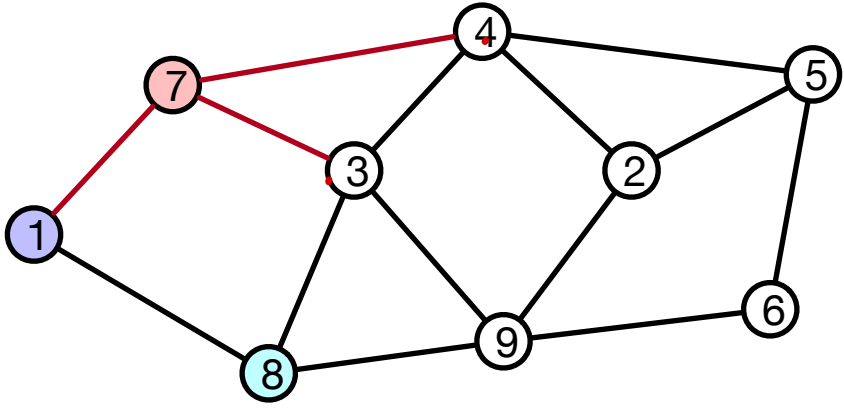
v	1	2	3	4	5	6	7	8	9
d[v]	0	-1	-1	-1	-1	-1	1	1	-1



cur

queue	7	8

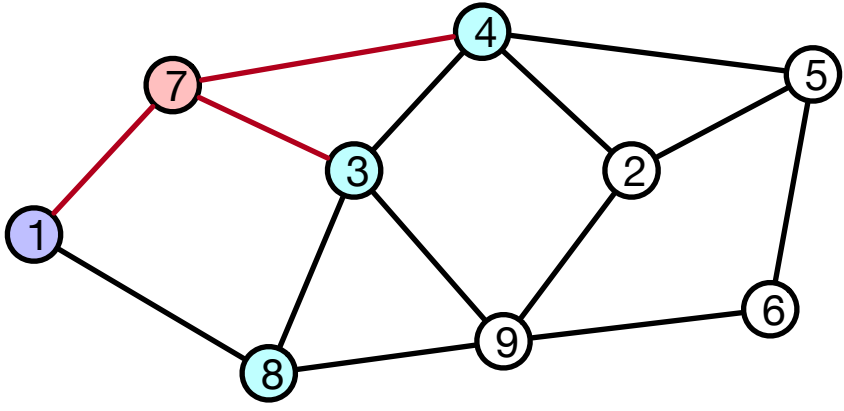
v	1	2	3	4	5	6	7	8	9
d[v]	0	-1	-1	-1	-1	-1	1	1	-1



cur	7

queue	8

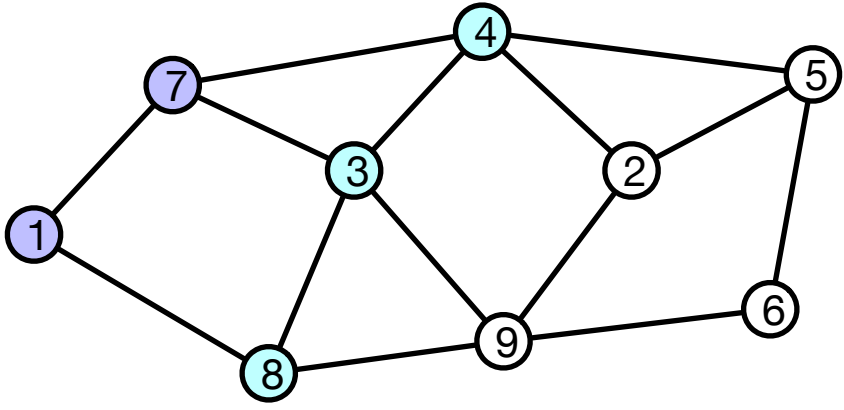
v	1	2	3	4	5	6	7	8	9
d[v]	0	-1	-1	-1	-1	-1	1	1	-1



cur	7

queue	8	3	4

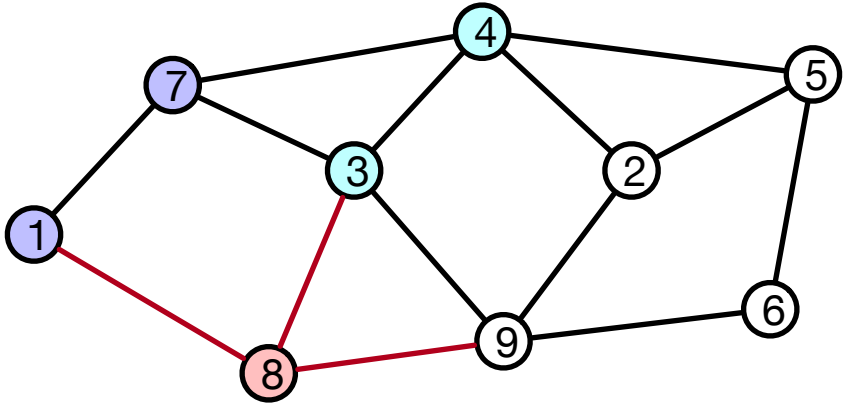
v	1	2	3	4	5	6	7	8	9
d[v]	0	-1	2	2	-1	-1	1	1	-1



cur

queue	8	3	4

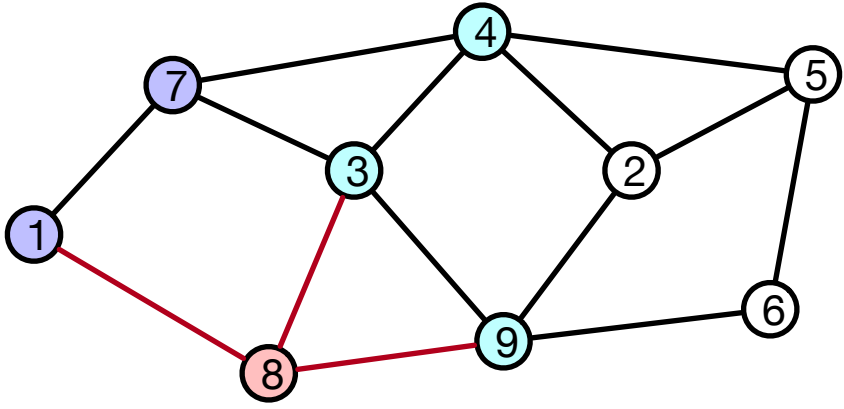
v	1	2	3	4	5	6	7	8	9
d[v]	0	-1	2	2	-1	-1	1	1	-1



cur	8

queue	3	4

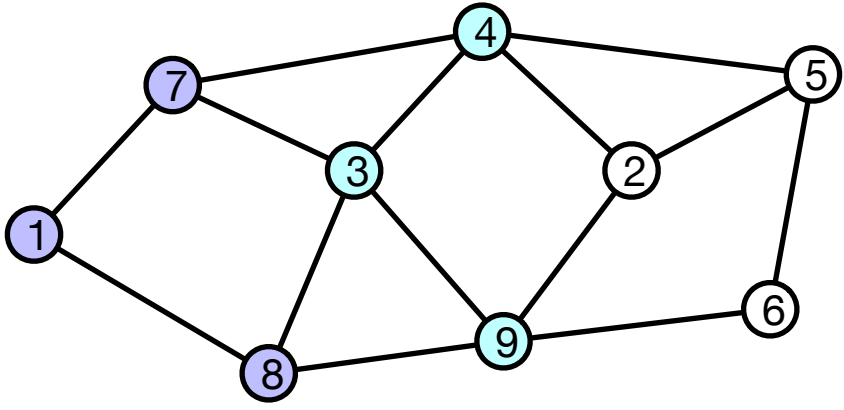
v	1	2	3	4	5	6	7	8	9
d[v]	0	-1	2	2	-1	-1	1	1	-1



cur	8

queue	3	4	9

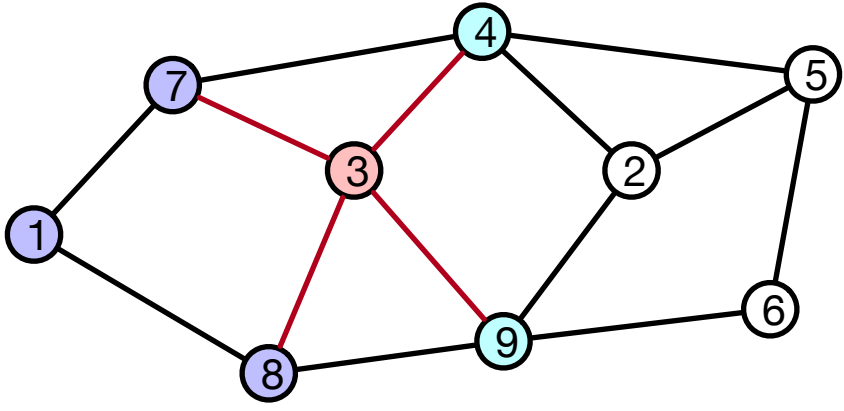
v	1	2	3	4	5	6	7	8	9
d[v]	0	-1	2	2	-1	-1	1	1	2



cur

queue	3	4	9

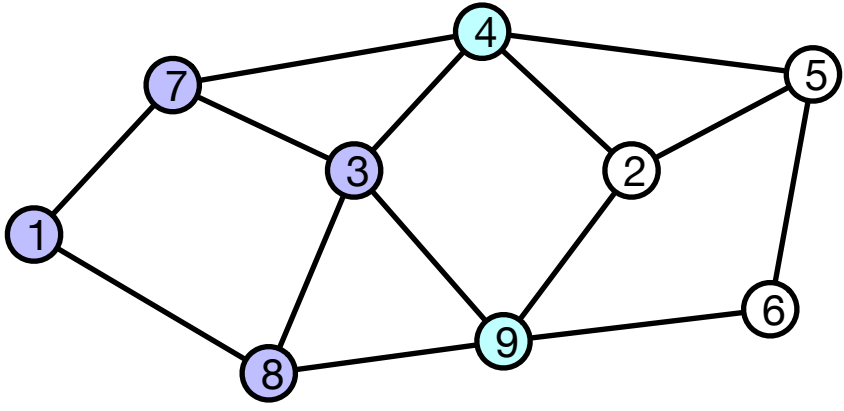
v	1	2	3	4	5	6	7	8	9
d[v]	0	-1	2	2	-1	-1	1	1	2



cur	3

queue	4	9

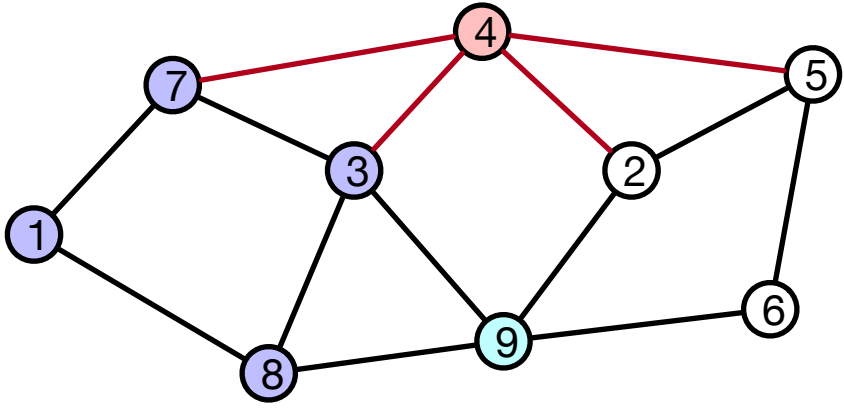
v	1	2	3	4	5	6	7	8	9
d[v]	0	-1	2	2	-1	-1	1	1	2



cur

queue	4	9

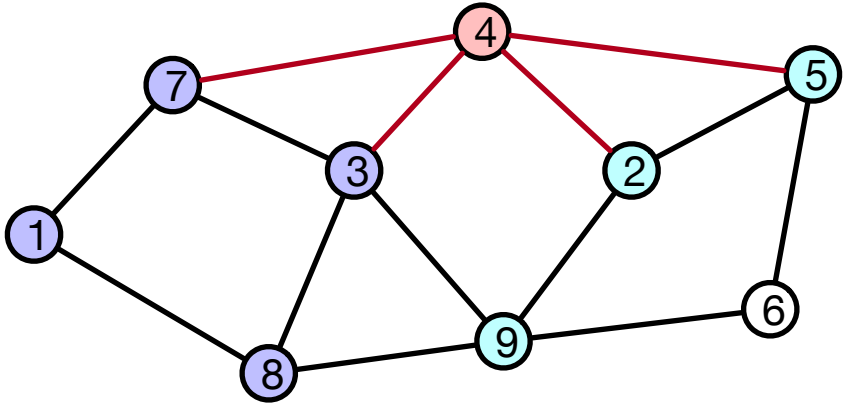
v	1	2	3	4	5	6	7	8	9
d[v]	0	-1	2	2	-1	-1	1	1	2



cur	4

queue	9

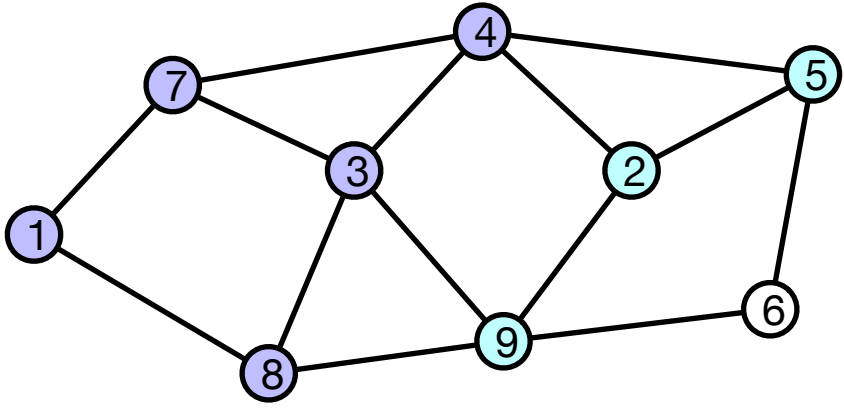
v	1	2	3	4	5	6	7	8	9
d[v]	0	-1	2	2	-1	-1	1	1	2



cur	4

queue	9	2	5

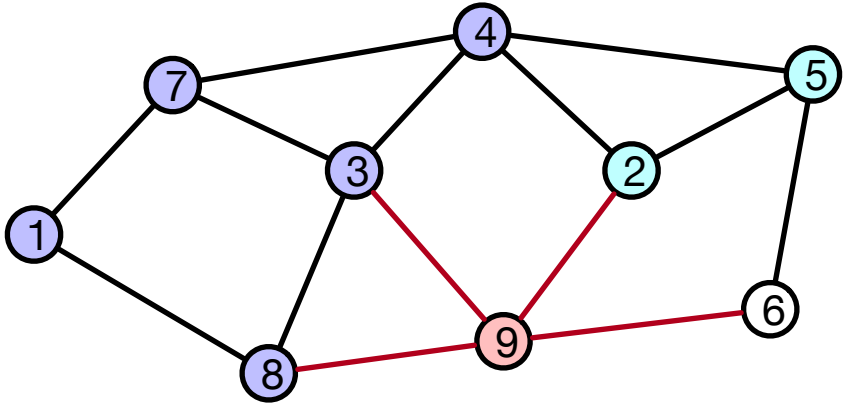
v	1	2	3	4	5	6	7	8	9
d[v]	0	3	2	2	3	-1	1	1	2



cur

queue	9	2	5

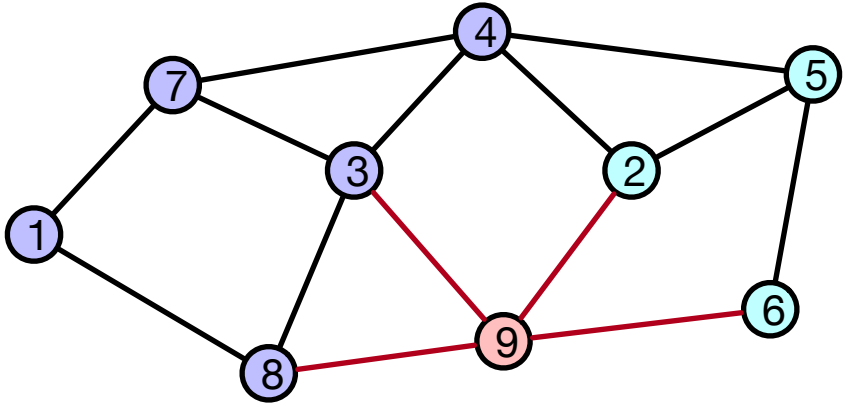
v	1	2	3	4	5	6	7	8	9
d[v]	0	3	2	2	3	-1	1	1	2



cur	9

queue	2	5

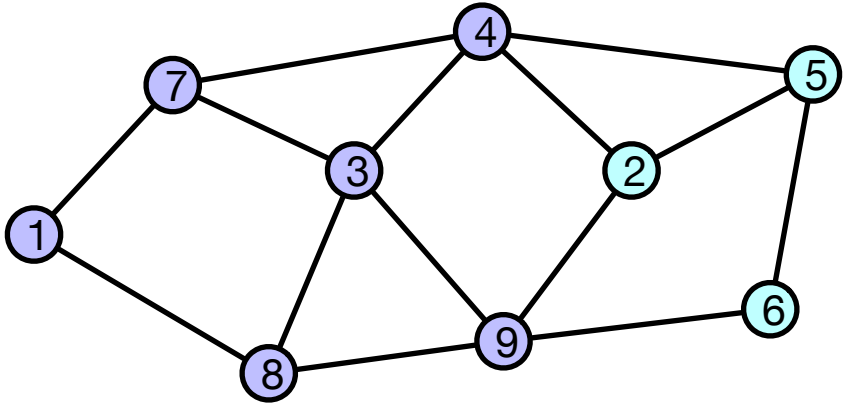
v	1	2	3	4	5	6	7	8	9
d[v]	0	3	2	2	3	-1	1	1	2



cur	9

queue	2	5	6

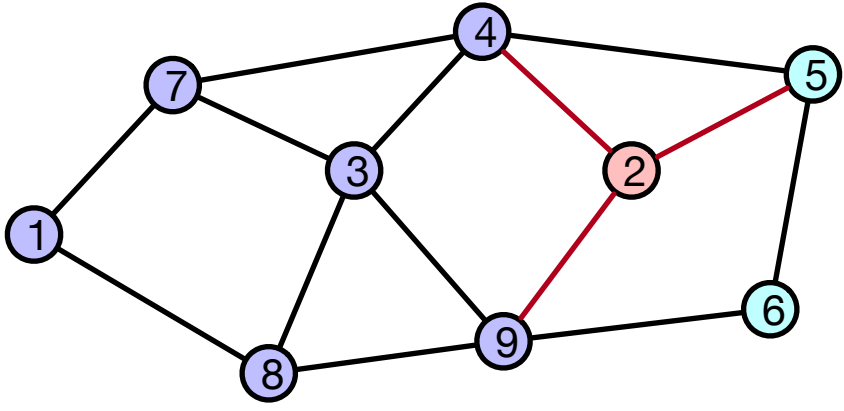
v	1	2	3	4	5	6	7	8	9
d[v]	0	3	2	2	3	3	1	1	2



cur

queue	2	5	6

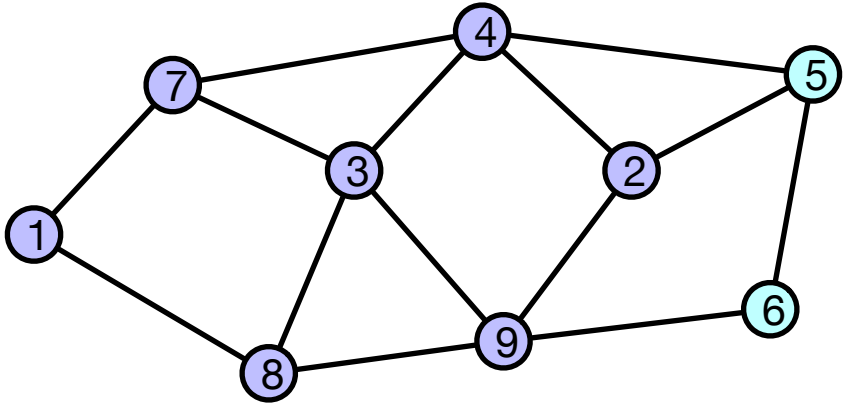
v	1	2	3	4	5	6	7	8	9
d[v]	0	3	2	2	3	3	1	1	2



cur	2

queue	5	6

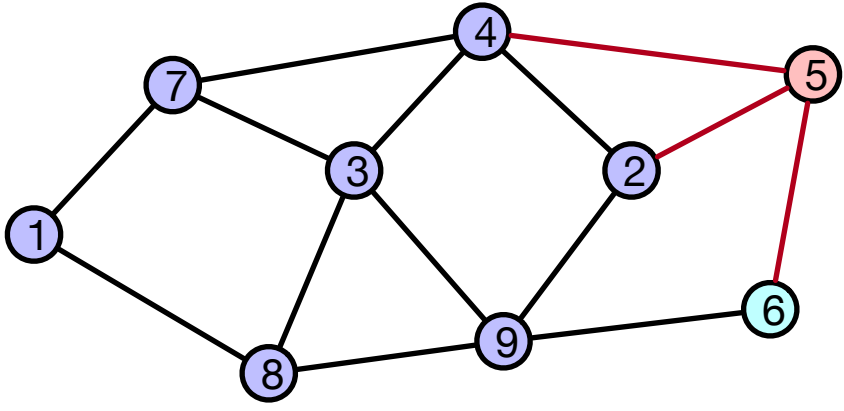
v	1	2	3	4	5	6	7	8	9
d[v]	0	3	2	2	3	3	1	1	2



cur

queue	5	6

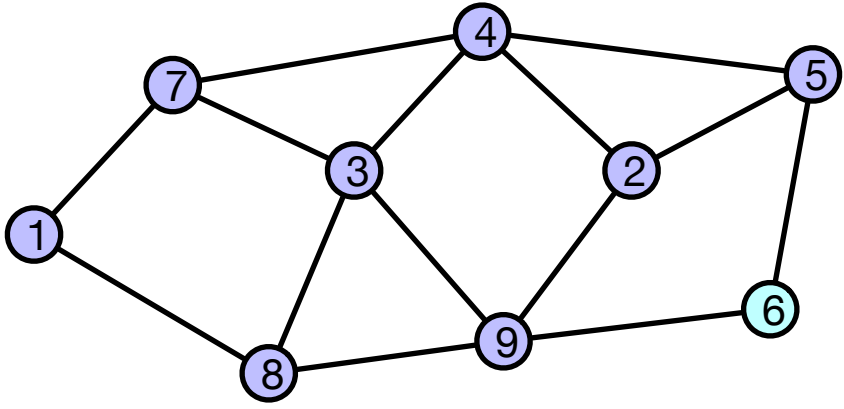
v	1	2	3	4	5	6	7	8	9
d[v]	0	3	2	2	3	3	1	1	2



cur	5

queue	6

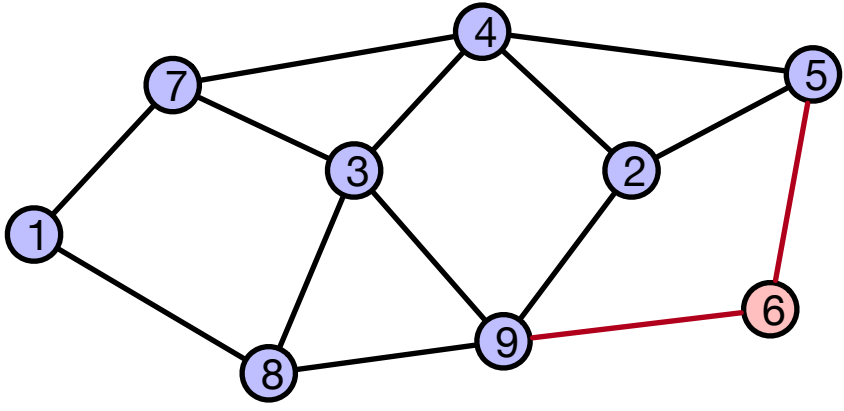
v	1	2	3	4	5	6	7	8	9
d[v]	0	3	2	2	3	3	1	1	2



cur

queue	6

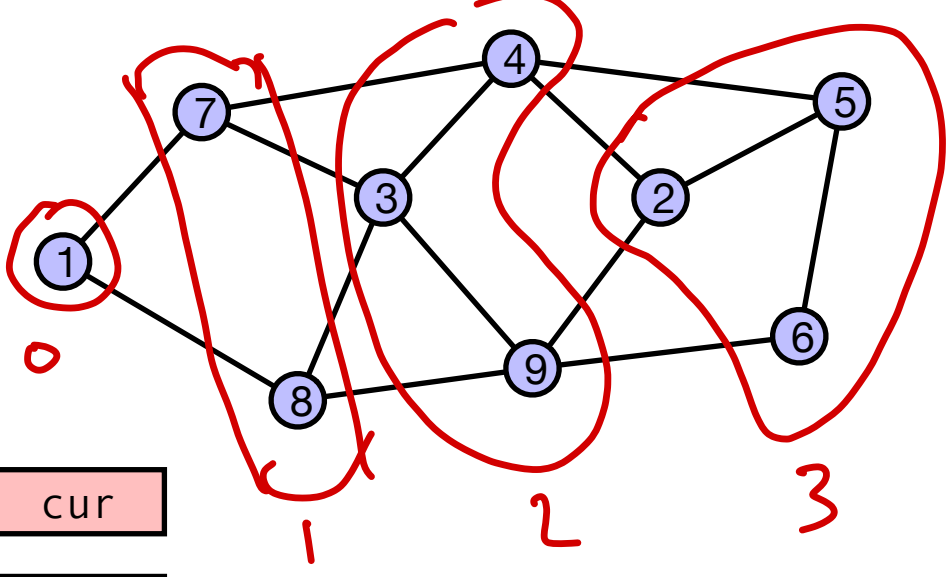
v	1	2	3	4	5	6	7	8	9
d[v]	0	3	2	2	3	3	1	1	2



cur	6

queue

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



cur

queue

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

BFS Correctness

Theorem. When `BFS(V, E, u)` terminates, for every vertex $v \in V$, $d[v]$ stores the distance (minimum number of hops) from u to v.

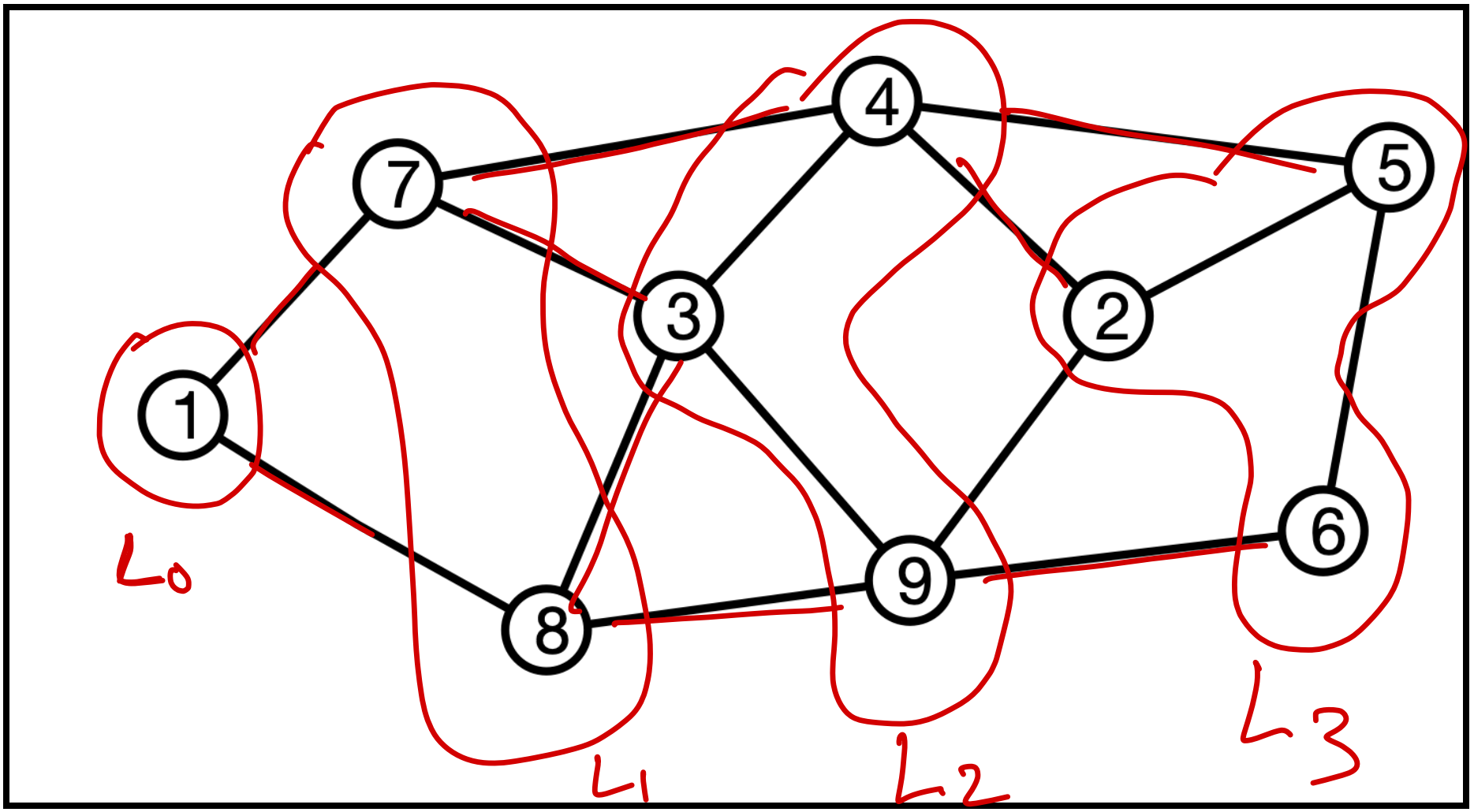
BFS Correctness

Theorem. When `BFS(V, E, u)` terminates, for every vertex $v \in V$, $d[v]$ stores the distance (minimum number of hops) from u to v.

Analysis. Break V into *layers*

- L_0 contains only u
- L_1 contains neighbors of u
- L_2 contains neighbors of neighbors of u
- $\vdots$
- L_k contains vertices not in $L_0, \ldots, L_{k-1}$ but with at least one neighbor in L_{k-1}

Layered Illustration



More Formally

For $i = 0, 1, 2, \ldots,$ Define L_i by

- $L_0 = \{u\}$
- $L_i =$ vertices not in $L_0, L_1, \ldots, L_{i-1}$ that have at least one neighbor in L_{i-1}

More Formally

For $i = 0, 1, 2, \ldots,$ Define L_i by

- $L_0 = \{u\}$
- $L_i =$ vertices not in $L_0, L_1, \ldots, L_{i-1}$ that have at least one neighbor in L_{i-1}

Claim. L_i contains precisely the vertices in V at distance i from u.

Prove by induction on i

Analysis of BFS

To Show

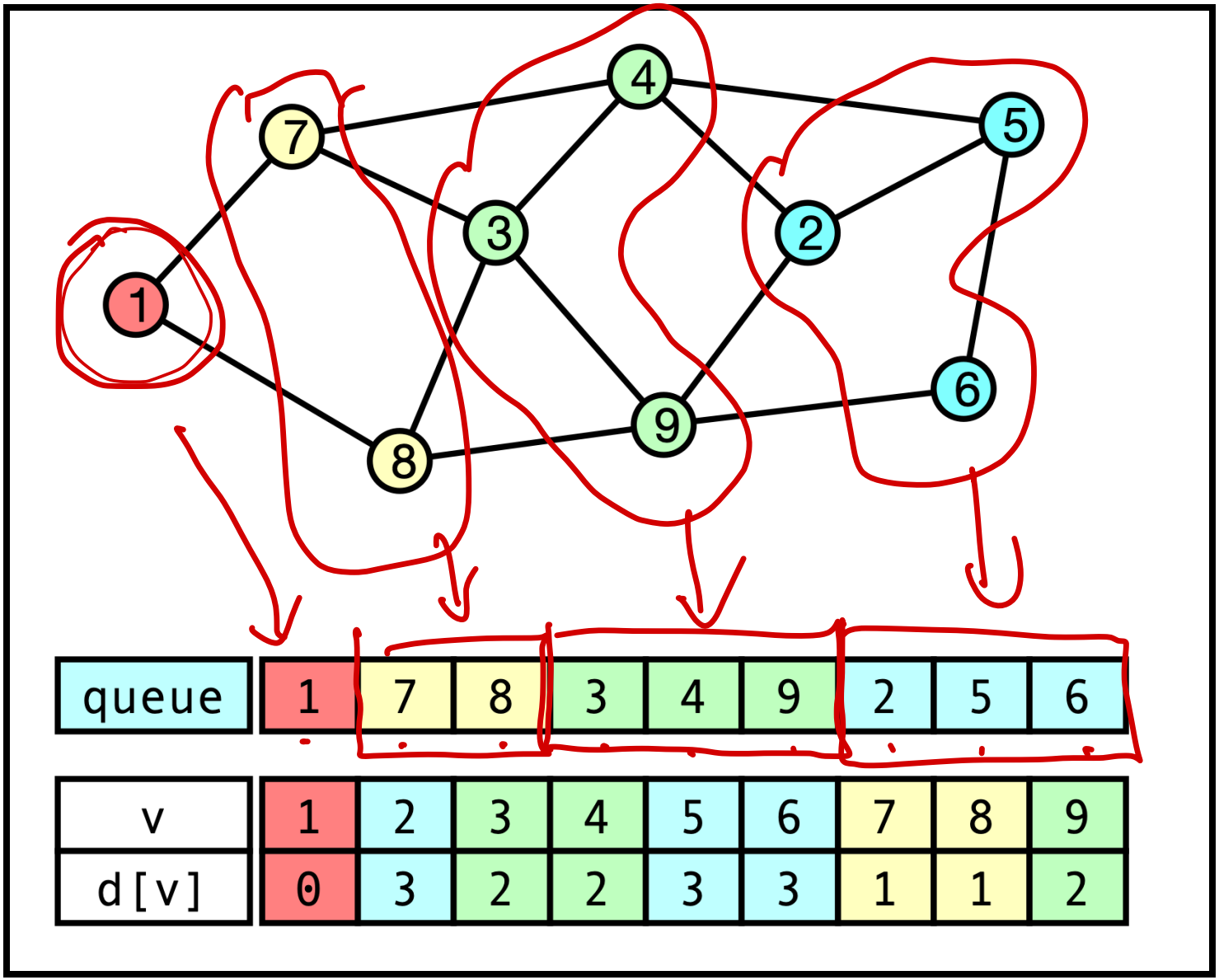
1. procedure finds vertices in increasing order of distance
2. distances are correctly computed when vertex is found (added to queue)

Idea. Break execution of BFS into *phases*

- phase i starts when first element of L_i is added to queue
- phase i ends when last element in L_i is added to queue

Phase Illustration

(1) distance
(2) layers
(3) Phases



Phase Claim

Claim. Consider an execution of BFS procedure. Then for every phase i:

1. phase i ends before phase $i + 1$ begins
2. every vertex from L_i is added to the queue in phase i
3. each vertex v added in phase i has $d[v] = i$

all elts in L_i added to queue before any in L_{i+1}

↑ nothing from L_i is missed

- If v in L_i then $d[v] = i$
- Previous claim was that if v in L_i then $d(u,v) = i$ ⇒ alg is correct

Phase Claim

Claim. Consider an execution of BFS procedure. Then for every phase i:

1. phase i ends before phase $i + 1$ begins
2. every vertex from L_i is added to the queue in phase i
3. each vertex v added in phase i has $d[v] = i$

Proof. Use induction on i

Base case $i = 0$. u is the only element in L_0, and it is added before any other elements, and $d[u]$ is initialized to 0.

Inductive Step of Phase Claim

To show: claim holds for $i+1$

Suppose claim holds for $j \leq i$ (inductive hypothesis). Then:

- when phase i ends (1) all vertices from L_i are in queue and (2) no vertex in L_{i+1} is in queue
- start removing elements in L_i from queue
- when v in L_i is removed, any neighbors in L_{i+1} are added to queue (if not already)
- distance is set to $d[v] \leftarrow i + 1$ ← $d[v] + 1 = i$ by ind. hyp.
- every v in L_{i+1} has neighbor in L_i
  - $\Longrightarrow$ all $v \in L_{i+1}$ are added to queue when last L_i vertex is removed from queue
- no vertex in L_{i+2} added to queue to this point

$\Rightarrow$ claim holds for phase i.

Conclusion

```
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
```

BFS procedure correctly computes all distances from u!

G: has n vertices, m edges.

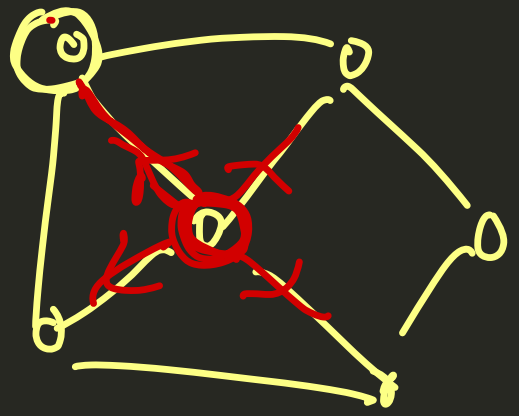
What is Running Time of BFS? — $O(m+n)$

```
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
```

$\Theta(n)$

$O(m)$ time

$O(1)$



What is running time of add/remove for queue? $O(1)$ ← true for array/linked list implementations

each vtx added to queue at most once

→ $O(m \cdot n)$ ~~~ $O(m+n)$

Consider removing $\underline{v}$ from queue

→ examine neighbors

$O(1)$ work per neighbor

→ $O(\deg(v))$

$\leq n$

Total work: $v_1, v_2, \ldots, v_n$

$$O(\deg(v_1)) + O(\deg(v_2)) + \cdots + O(\deg(v_n))$$

$$= O(\underbrace{\deg(v_1) + \deg(v_2) + \cdots + \deg(v_n)}_{2 \cdot m})$$