# Breadth-first search COMS20010 2020, Video 4-3

John Lapinskas, University of Bristol

# **Shortest** path-finding

**Last time:** Given a graph G and two vertices  $x, y \in V(G)$ , is there a path from x to y?

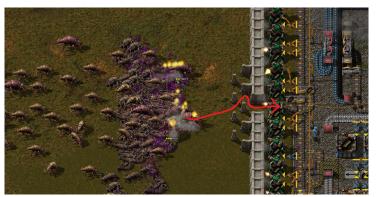
E.g. can an enemy attack the base without breaking down a wall?



# **Shortest** path-finding

**Last time:** Given a graph G and two vertices  $x, y \in V(G)$ , is there a path from x to y?

E.g. can an enemy attack the base without breaking down a wall?



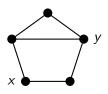
**This time:** What is the **shortest** path from x to y?

**This time:** What is the **shortest** path from x to y?

What do we mean by "shortest"?

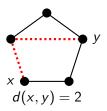
**This time:** What is the **shortest** path from x to y?

What do we mean by "shortest"?



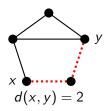
**This time:** What is the **shortest** path from x to y?

What do we mean by "shortest"?



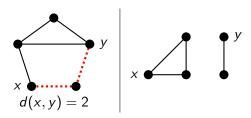
**This time:** What is the **shortest** path from x to y?

What do we mean by "shortest"?



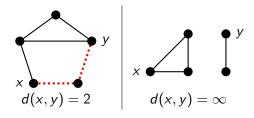
**This time:** What is the **shortest** path from x to y?

What do we mean by "shortest"?



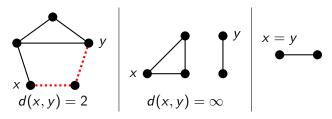
**This time:** What is the **shortest** path from x to y?

What do we mean by "shortest"?



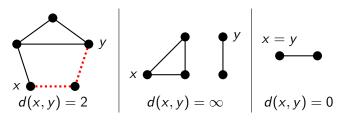
**This time:** What is the **shortest** path from x to y?

What do we mean by "shortest"?



**This time:** What is the **shortest** path from x to y?

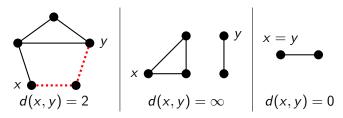
What do we mean by "shortest"?



**This time:** What is the **shortest** path from x to y?

What do we mean by "shortest"?

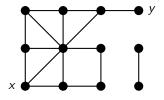
The **distance** between x and y, d(x, y), is the length in edges of a shortest path between x and y, or  $\infty$  if no such path exists.



In directed graphs, it's the same except that the path is **from** x **to** y. So... we might not have d(x,y) = d(y,x)!

**Input:** A graph G and two vertices x and y.

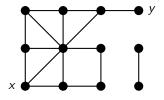
**Output:** A shortest path from x to y.



Let  $L_i$  be the set of vertices at distance i from x.

**Input:** A graph G and two vertices x and y.

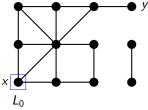
**Output:** A shortest path from x to y.



Let  $L_i$  be the set of vertices at distance i from x. So  $L_0 = \{x\}$ .

**Input:** A graph G and two vertices x and y.

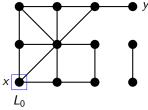
**Output:** A shortest path from x to y.



Let  $L_i$  be the set of vertices at distance i from x. So  $L_0 = \{x\}$ .

**Input:** A graph G and two vertices x and y.

**Output:** A shortest path from x to y.

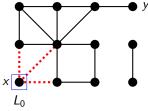


Let  $L_i$  be the set of vertices at distance i from x. So  $L_0 = \{x\}$ .

 $L_1$  is everything adjacent to x.

**Input:** A graph G and two vertices x and y.

**Output:** A shortest path from x to y.

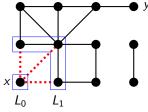


Let  $L_i$  be the set of vertices at distance i from x. So  $L_0 = \{x\}$ .

 $L_1$  is everything adjacent to x.

**Input:** A graph G and two vertices x and y.

**Output:** A shortest path from x to y.

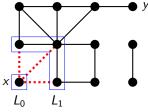


Let  $L_i$  be the set of vertices at distance i from x. So  $L_0 = \{x\}$ .

 $L_1$  is everything adjacent to x.

**Input:** A graph G and two vertices x and y.

**Output:** A shortest path from x to y.



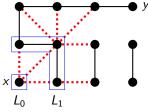
Let  $L_i$  be the set of vertices at distance i from x. So  $L_0 = \{x\}$ .

 $L_1$  is everything adjacent to x.

 $L_2$  is everything adjacent to  $L_1$ , but **not** in  $L_0$  or  $L_1$ .

**Input:** A graph G and two vertices x and y.

**Output:** A shortest path from x to y.



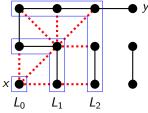
Let  $L_i$  be the set of vertices at distance i from x. So  $L_0 = \{x\}$ .

 $L_1$  is everything adjacent to x.

 $L_2$  is everything adjacent to  $L_1$ , but **not** in  $L_0$  or  $L_1$ .

**Input:** A graph G and two vertices x and y.

**Output:** A shortest path from x to y.



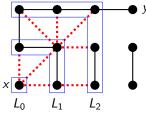
Let  $L_i$  be the set of vertices at distance i from x. So  $L_0 = \{x\}$ .

 $L_1$  is everything adjacent to x.

 $L_2$  is everything adjacent to  $L_1$ , but **not** in  $L_0$  or  $L_1$ .

**Input:** A graph G and two vertices x and y.

**Output:** A shortest path from x to y.



Let  $L_i$  be the set of vertices at distance i from x. So  $L_0 = \{x\}$ .

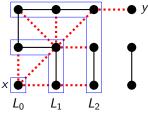
 $L_1$  is everything adjacent to x.

 $L_2$  is everything adjacent to  $L_1$ , but **not** in  $L_0$  or  $L_1$ .

In general,  $L_{i+1}$  is everything adjacent to  $L_i$  and not in  $L_0 \cup \cdots \cup L_i$ .

**Input:** A graph G and two vertices x and y.

**Output:** A shortest path from x to y.



Let  $L_i$  be the set of vertices at distance i from x. So  $L_0 = \{x\}$ .

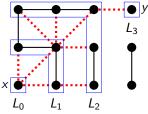
 $L_1$  is everything adjacent to x.

 $L_2$  is everything adjacent to  $L_1$ , but **not** in  $L_0$  or  $L_1$ .

In general,  $L_{i+1}$  is everything adjacent to  $L_i$  and not in  $L_0 \cup \cdots \cup L_i$ .

**Input:** A graph G and two vertices x and y.

**Output:** A shortest path from x to y.



Let  $L_i$  be the set of vertices at distance i from x. So  $L_0 = \{x\}$ .

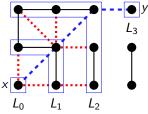
 $L_1$  is everything adjacent to x.

 $L_2$  is everything adjacent to  $L_1$ , but **not** in  $L_0$  or  $L_1$ .

In general,  $L_{i+1}$  is everything adjacent to  $L_i$  and not in  $L_0 \cup \cdots \cup L_i$ .

**Input:** A graph G and two vertices x and y.

**Output:** A shortest path from x to y.

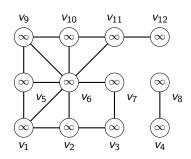


Let  $L_i$  be the set of vertices at distance i from x. So  $L_0 = \{x\}$ .

 $L_1$  is everything adjacent to x.

 $L_2$  is everything adjacent to  $L_1$ , but **not** in  $L_0$  or  $L_1$ .

In general,  $L_{i+1}$  is everything adjacent to  $L_i$  and not in  $L_0 \cup \cdots \cup L_i$ .



queue:

#### Algorithm: BFS

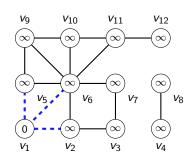
: Graph G = (V, E), vertex  $v \in V$ . Input **Output** : d(v, y) for all  $y \in V$  and "a way of finding shortest paths". 1 Number the vertices of G as  $v = v_1, \dots, v_n$ . 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ . 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ . 4 Let queue be a queue containing all tuples

 $(v, v_i)$  with  $\{v, v_i\} \in E$ .

5 while queue is not empty do

Remove front tuple  $(v_i, v_i)$  from queue. if  $L[i] = \infty$  then

Add  $(v_i, v_k)$  to queue for all  $\{v_j,v_k\}\in E,\ k\neq i.$ Set  $L[j] \leftarrow L[i] + 1$ , pred[j] = i. 9



queue: (1,5), (1,6), (1,2)

#### Algorithm: BFS

```
Input : Graph G = (V, E), vertex v \in V.

Output : d(v, y) for all y \in V and "a way of finding shortest paths".

1 Number the vertices of G as v = v_1, \dots, v_n.

2 Let L[i] \leftarrow \infty for all i \in [n].

3 Let L[i] \leftarrow 0, pred[i] \leftarrow None.

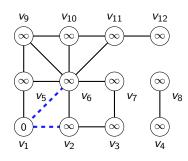
4 Let queue be a queue containing all tuples (v, v_i) with \{v, v_i\} \in E.
```

5 while queue is not empty do

Remove front tuple  $(v_i, v_j)$  from queue.

If  $L[j] = \infty$  then

Add  $(v_j, v_k)$  to queue for all



queue: (1,6),(1,2)

#### Algorithm: BFS

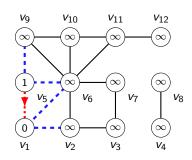
: Graph G = (V, E), vertex  $v \in V$ . Input **Output** : d(v, y) for all  $y \in V$  and "a way of finding shortest paths". 1 Number the vertices of G as  $v = v_1, \dots, v_n$ . 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ . 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ .

4 Let queue be a queue containing all tuples  $(v, v_i)$  with  $\{v, v_i\} \in E$ .

5 while queue is not empty do

Remove front tuple  $(v_i, v_i)$  from queue. if  $L[i] = \infty$  then

Add  $(v_i, v_k)$  to queue for all  $\{v_j,v_k\}\in E,\ k\neq i.$ Set  $L[j] \leftarrow L[i] + 1$ , pred[j] = i. 9



queue: (1,6),(1,2),(5,9),(5,6)

#### Algorithm: BFS

```
Input : Graph G = (V, E), vertex v \in V.

Output : d(v, y) for all y \in V and "a way of finding shortest paths".
```

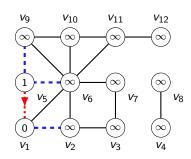
- 1 Number the vertices of G as  $v = v_1, \dots, v_n$ .
- 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ .
- 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ .
- 4 Let queue be a queue containing all tuples  $(v, v_j)$  with  $\{v, v_j\} \in E$ .
- 5 while queue is not empty do

Remove front tuple 
$$(v_i, v_j)$$
 from queue.

If  $L[j] = \infty$  then

Add  $(v_j, v_k)$  to queue for all  $\{v_j, v_k\} \in E, k \neq i$ .

Set  $L[j] \leftarrow L[j] + 1$ , pred $[j] = i$ .



queue: (1,2), (5,9), (5,6)

#### Algorithm: BFS

```
Input : Graph G = (V, E), vertex v ∈ V.

Output : d(v, y) for all y ∈ V and "a way of finding shortest paths".

1 Number the vertices of G as v = v<sub>1</sub>,..., v<sub>n</sub>.

2 Let L[i] ← ∞ for all i ∈ [n].

3 Let L[I] ← 0, pred[I] ← None.

4 Let queue be a queue containing all tuples
```

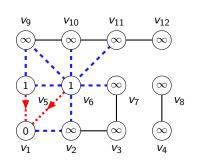
4 Let queue be a queue containing all tuples (v, v<sub>j</sub>) with {v, v<sub>j</sub>} ∈ E.
 5 while queue is not empty do

Remove front tuple  $(v_i, v_j)$  from queue.

If  $L[j] = \infty$  then

Add  $(v_j, v_k)$  to queue for all  $\{v_j, v_k\} \in E, k \neq i$ .

Set  $L[j] \leftarrow L[j] + 1$ , pred[j] = i.

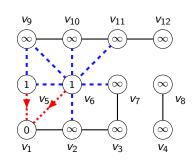


queue: 
$$(1,2), (5,9), (5,6), (6,5),$$
  
 $(6,9), (6,10), (6,11), (6,7),$   
 $(6,2)$ 

#### Algorithm: BFS

```
: Graph G = (V, E), vertex v \in V.
  Input
              : d(v, y) for all y \in V and "a way of
                finding shortest paths".
 Number the vertices of G as v = v_1, \dots, v_n.
2 Let L[i] \leftarrow \infty for all i \in [n].
3 Let L[1] \leftarrow 0, pred[1] \leftarrow None.
4 Let queue be a queue containing all tuples
    (v, v_i) with \{v, v_i\} \in E.
5 while queue is not empty do
        Remove front tuple (v_i, v_i) from queue.
        if L[j] = \infty then
             Add (v_i, v_k) to queue for all
              \{v_i, v_k\} \in E, k \neq i.
             Set L[i] \leftarrow L[i] + 1, pred[i] = i.
```

Return L and pred.



$$\begin{array}{l} \text{queue:} \, (5,9), (5,6), (6,5), (6,9), \\ (6,10), (6,11), (6,7), (6,2), \end{array}$$

#### Algorithm: BFS

Input : Graph G = (V, E), vertex  $v \in V$ .

Output : d(v, y) for all  $y \in V$  and "a way of finding shortest paths".

1 Number the vertices of G as  $v = v_1, \ldots, v_n$ .

2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ .

3 Let  $L[i] \leftarrow 0$ , pred $[i] \leftarrow None$ .

4 Let queue be a queue containing all tuples  $(v, v_j)$  with  $\{v, v_j\} \in E$ .

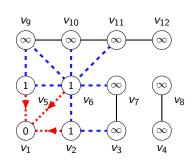
5 while queue is not empty do

Remove front tuple  $(v_i, v_j)$  from queue. if  $L[j] = \infty$  then

Add  $(v_i, v_k)$  to queue for all

Add 
$$(v_j, v_k)$$
 to queue for all  $\{v_j, v_k\} \in E, \ k \neq i.$ 
Set  $L[j] \leftarrow L[j] + 1$ ,  $pred[j] = i$ .

10 Return L and pred.

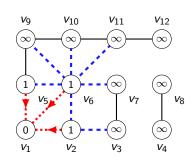


$$\begin{array}{c} \mathtt{queue:} \, (5,9), (5,6), (6,5), (6,9), \\ (6,10), (6,11), (6,7), (6,2), \\ (2,6), (2,3) \end{array}$$

#### Algorithm: BFS

: Graph G = (V, E), vertex  $v \in V$ . Input : d(v, y) for all  $y \in V$  and "a way of finding shortest paths". Number the vertices of G as  $v = v_1, \dots, v_n$ . 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ . 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ . 4 Let queue be a queue containing all tuples  $(v, v_i)$  with  $\{v, v_i\} \in E$ . 5 while queue is not empty do Remove front tuple  $(v_i, v_i)$  from queue. if  $L[j] = \infty$  then Add  $(v_i, v_k)$  to queue for all  $\{v_i, v_k\} \in E, k \neq i.$ Set  $L[i] \leftarrow L[i] + 1$ , pred[i] = i.

Return L and pred.

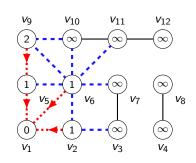


queue: 
$$(5,6), (6,5), (6,9), (6,10),$$
  
 $(6,11), (6,7), (6,2), (2,6),$   
 $(2,3)$ 

#### Algorithm: BFS

: Graph G = (V, E), vertex  $v \in V$ . Input : d(v, y) for all  $y \in V$  and "a way of finding shortest paths". Number the vertices of G as  $v = v_1, \dots, v_n$ . 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ . 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ . 4 Let queue be a queue containing all tuples  $(v, v_i)$  with  $\{v, v_i\} \in E$ . 5 while queue is not empty do Remove front tuple  $(v_i, v_i)$  from queue. if  $L[j] = \infty$  then Add  $(v_i, v_k)$  to queue for all  $\{v_i, v_k\} \in E, k \neq i.$ Set  $L[i] \leftarrow L[i] + 1$ , pred[i] = i.

10 Return L and pred.



queue: 
$$(5,6), (6,5), (6,9), (6,10),$$
  
 $(6,11), (6,7), (6,2), (2,6),$   
 $(2,3), (9,10), (9,6)$ 

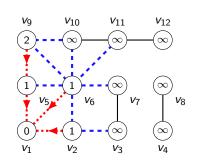
#### Algorithm: BFS

: Graph G = (V, E), vertex  $v \in V$ . Input : d(v, y) for all  $y \in V$  and "a way of finding shortest paths". Number the vertices of G as  $v = v_1, \dots, v_n$ . 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ . 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ . 4 Let queue be a queue containing all tuples  $(v, v_i)$  with  $\{v, v_i\} \in E$ . 5 while queue is not empty do Remove front tuple  $(v_i, v_i)$  from queue. if  $L[i] = \infty$  then Add  $(v_i, v_k)$  to queue for all  $\{v_i, v_k\} \in E, k \neq i.$ Set  $L[i] \leftarrow L[i] + 1$ , pred[i] = i.

10 Return L and pred.

q

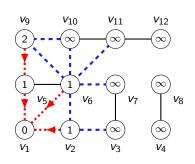
John Lapinskas Video 4-3 5/6



$$\begin{array}{l} \mathtt{queue:}\,(6,5),(6,9),(6,10),(6,11),\\ (6,7),(6,2),(2,6),(2,3),\\ (9,10),(9,6) \end{array}$$

#### Algorithm: BFS

: Graph G = (V, E), vertex  $v \in V$ . Input : d(v, y) for all  $y \in V$  and "a way of finding shortest paths". Number the vertices of G as  $v = v_1, \dots, v_n$ . 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ . 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ . 4 Let queue be a queue containing all tuples  $(v, v_i)$  with  $\{v, v_i\} \in E$ . 5 while queue is not empty do Remove front tuple  $(v_i, v_i)$  from queue. if  $L[i] = \infty$  then Add  $(v_i, v_k)$  to queue for all  $\{v_i, v_k\} \in E, k \neq i.$ Set  $L[i] \leftarrow L[i] + 1$ , pred[i] = i.



queue: 
$$(6,9)$$
,  $(6,10)$ ,  $(6,11)$ ,  $(6,7)$ ,  $(6,2)$ ,  $(2,6)$ ,  $(2,3)$ ,  $(9,10)$ ,  $(9,6)$ 

### Algorithm: BFS

Input : Graph G = (V, E), vertex  $v \in V$ .

Output : d(v, y) for all  $y \in V$  and "a way of finding shortest paths".

Number the vertices of G as  $v = v_1, \ldots, v_n$ .

Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ .

Let  $L[i] \leftarrow 0$ ,  $Pred[i] \leftarrow None$ .

Let queue be a queue containing all tuples  $(v, v_j)$  with  $\{v, v_j\} \in E$ .

while queue is not empty do

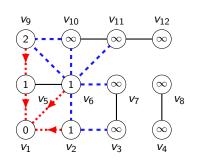
Remove front tuple  $(v_i, v_j)$  from queue.

Remove front tuple  $(v_i, v_j)$  from queue.  $L[i] = \infty$  then

Add  $L[i] = \infty$  then

Add  $L[i] = \infty$  then  $L[i] = \infty$  then

Set  $L[i] \leftarrow L[i] + 1$ , pred[i] = i.



queue: 
$$(6,10)$$
,  $(6,11)$ ,  $(6,7)$ ,  $(6,2)$ ,  $(2,6)$ ,  $(2,3)$ ,  $(9,10)$ ,  $(9,6)$ 

### Algorithm: BFS

Input : Graph G = (V, E), vertex  $v \in V$ .

Output : d(v, y) for all  $y \in V$  and "a way of finding shortest paths".

1 Number the vertices of G as  $v = v_1, \ldots, v_n$ .

2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ .

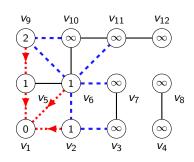
3 Let  $L[i] \leftarrow 0$ , pred $[i] \leftarrow N$ one.

4 Let queue be a queue containing all tuples  $(v, v_j)$  with  $\{v, v_j\} \in E$ .

5 while queue is not empty do

6 Remove front tuple  $(v_i, v_j)$  from queue.

7 If  $L[j] = \infty$  then



$$\begin{array}{c} \mathtt{queue:}\,(6,11),(6,7),(6,2),(2,6),\\ (2,3),(9,10),(9,6) \end{array}$$

### Algorithm: BFS

Input : Graph G = (V, E), vertex  $v \in V$ .

Output : d(v, y) for all  $y \in V$  and "a way of finding shortest paths".

1 Number the vertices of G as  $v = v_1, \ldots, v_n$ .

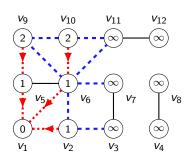
2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ .

3 Let  $L[I] \leftarrow 0$ , pred $[I] \leftarrow None$ .

3 Let L[I] ← 0, pred[I] ← None.
 4 Let queue be a queue containing all tuples (v, v<sub>i</sub>) with {v, v<sub>i</sub>} ∈ E.

5 while queue is not empty do

6 Remove front tuple  $(v_i, v_j)$  from queue. 7 if  $L[j] = \infty$  then 8 Add  $(v_j, v_k)$  to queue for all  $\{v_j, v_k\} \in E, k \neq i$ . 9 Set  $L[j] \leftarrow L[j] + 1$ , pred[j] = i.

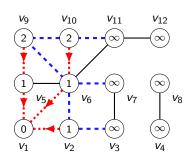


queue: 
$$(6,11)$$
,  $(6,7)$ ,  $(6,2)$ ,  $(2,6)$ ,  $(2,3)$ ,  $(9,10)$ ,  $(9,6)$ ,  $(10,11)$ ,  $(10,9)$ 

### Algorithm: BFS

: Graph G = (V, E), vertex  $v \in V$ . Input : d(v, y) for all  $y \in V$  and "a way of finding shortest paths". Number the vertices of G as  $v = v_1, \dots, v_n$ . 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ . 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ . 4 Let queue be a queue containing all tuples  $(v, v_i)$  with  $\{v, v_i\} \in E$ . 5 while queue is not empty do Remove front tuple  $(v_i, v_i)$  from queue. if  $L[j] = \infty$  then

Add  $(v_i, v_k)$  to queue for all  $\{v_i, v_k\} \in E, k \neq i.$ Set  $L[i] \leftarrow L[i] + 1$ , pred[i] = i.



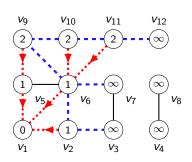
$$\begin{array}{c} \text{queue:} \, (6,7), (6,2), (2,6), (2,3), \\ (9,10), (9,6), (10,11), (10,9) \end{array}$$

### Algorithm: BFS

: Graph G = (V, E), vertex  $v \in V$ . Input Output : d(v, y) for all  $y \in V$  and "a way of finding shortest paths". Number the vertices of G as  $v = v_1, \dots, v_n$ .

- 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ .
- 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ .
- 4 Let queue be a queue containing all tuples  $(v, v_i)$  with  $\{v, v_i\} \in E$ .
- 5 while queue is not empty do

Remove front tuple  $(v_i, v_i)$  from queue. if  $L[j] = \infty$  then Add  $(v_i, v_k)$  to queue for all  $\{v_i, v_k\} \in E, k \neq i.$ Set  $L[i] \leftarrow L[i] + 1$ , pred[i] = i.



queue: 
$$(6,7)$$
,  $(6,2)$ ,  $(2,6)$ ,  $(2,3)$ ,  $(9,10)$ ,  $(9,6)$ ,  $(10,11)$ ,  $(10,9)$ ,  $(11,10)$ ,  $(11,12)$ 

### Algorithm: BFS

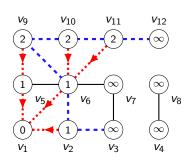
: Graph G = (V, E), vertex  $v \in V$ . Input : d(v, y) for all  $y \in V$  and "a way of Output finding shortest paths". Number the vertices of G as  $v = v_1, \dots, v_n$ . 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ . 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ . 4 Let queue be a queue containing all tuples  $(v, v_i)$  with  $\{v, v_i\} \in E$ . 5 while queue is not empty do Remove front tuple  $(v_i, v_i)$  from queue. if  $L[j] = \infty$  then Add  $(v_i, v_k)$  to queue for all

 $\{v_i, v_k\} \in E, k \neq i.$ 

Set  $L[i] \leftarrow L[i] + 1$ , pred[i] = i.

Return L and pred.

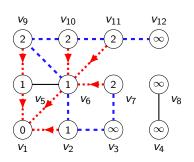
John Lapinskas Video 4-3 5/6



queue: 
$$(6,2)$$
,  $(2,6)$ ,  $(2,3)$ ,  $(9,10)$ ,  $(9,6)$ ,  $(10,11)$ ,  $(10,9)$ ,  $(11,10)$ ,  $(11,12)$ 

### Algorithm: BFS

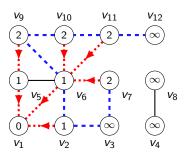
```
: Graph G = (V, E), vertex v \in V.
  Input
              : d(v, y) for all y \in V and "a way of
  Output
                finding shortest paths".
1 Number the vertices of G as v = v_1, \dots, v_n.
2 Let L[i] \leftarrow \infty for all i \in [n].
3 Let L[1] \leftarrow 0, pred[1] \leftarrow None.
4 Let queue be a queue containing all tuples
    (v, v_i) with \{v, v_i\} \in E.
5 while queue is not empty do
        Remove front tuple (v_i, v_i) from queue.
        if L[i] = \infty then
             Add (v_i, v_k) to queue for all
              \{v_i, v_k\} \in E, k \neq i.
             Set L[i] \leftarrow L[i] + 1, pred[i] = i.
  Return L and pred.
```



queue: 
$$(6,2), (2,6), (2,3), (9,10),$$
  $_{9}$   $\begin{bmatrix} \{v_j, v_k\} \\ \text{Set L}[j] \leftarrow \\ (9,6), (10,11), (10,9), (11,10), \\ (11,12), (7,3) \end{bmatrix}$  Return L and pred.

### Algorithm: BFS

```
: Graph G = (V, E), vertex v \in V.
  Input
              : d(v, y) for all y \in V and "a way of
                finding shortest paths".
 Number the vertices of G as v = v_1, \dots, v_n.
2 Let L[i] \leftarrow \infty for all i \in [n].
3 Let L[1] \leftarrow 0, pred[1] \leftarrow None.
4 Let queue be a queue containing all tuples
    (v, v_i) with \{v, v_i\} \in E.
5 while queue is not empty do
        Remove front tuple (v_i, v_i) from queue.
        if L[i] = \infty then
             Add (v_i, v_k) to queue for all
              \{v_i, v_k\} \in E, k \neq i.
             Set L[j] \leftarrow L[i] + 1, pred[j] = i.
```



queue: 
$$(2,6), (2,3), (9,10), (9,6),$$
 g  $\begin{bmatrix} \{v_j, v_k\} \\ \text{Set L}[j] \leftarrow \\ (7,3) \end{bmatrix}$  10 Return L and pred.

### Algorithm: BFS

```
Input : Graph G = (V, E), vertex v \in V.

Output : d(v, y) for all y \in V and "a way of finding shortest paths".

1 Number the vertices of G as v = v_1, \ldots, v_n.

2 Let L[i] \leftarrow \infty for all i \in [n].

3 Let L[i] \leftarrow 0, pred[i] \leftarrow None.

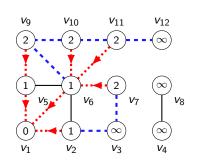
4 Let queue be a queue containing all tuples (v, v_j) with \{v, v_j\} \in E.

5 while queue is not empty do

6 Remove front tuple (v_i, v_j) from queue.

7 If L[j] = \infty then
```

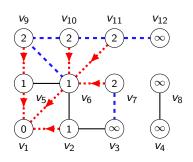
John Lapinskas Video 4-3 5 / 6



queue: 
$$(2,3), (9,10), (9,6), (10,11), _{9}$$
  
 $(10,9), (11,10), (11,12), (7,3)$ 

### Algorithm: BFS

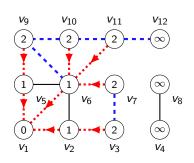
```
: Graph G = (V, E), vertex v \in V.
  Input
              : d(v, y) for all y \in V and "a way of
  Output
                finding shortest paths".
 Number the vertices of G as v = v_1, \dots, v_n.
2 Let L[i] \leftarrow \infty for all i \in [n].
3 Let L[1] \leftarrow 0, pred[1] \leftarrow None.
4 Let queue be a queue containing all tuples
    (v, v_i) with \{v, v_i\} \in E.
5 while queue is not empty do
        Remove front tuple (v_i, v_i) from queue.
        if L[i] = \infty then
             Add (v_i, v_k) to queue for all
              \{v_i, v_k\} \in E, k \neq i.
             Set L[j] \leftarrow L[i] + 1, pred[j] = i.
```



queue: 
$$(9,10)$$
,  $(9,6)$ ,  $(10,11)$ ,  $(10,9)_9$   
 $(11,10)$ ,  $(11,12)$ ,  $(7,3)$ 

### Algorithm: BFS

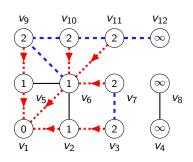
```
: Graph G = (V, E), vertex v \in V.
  Input
              : d(v, y) for all y \in V and "a way of
  Output
                finding shortest paths".
1 Number the vertices of G as v = v_1, \ldots, v_n.
2 Let L[i] \leftarrow \infty for all i \in [n].
3 Let L[1] \leftarrow 0, pred[1] \leftarrow None.
4 Let queue be a queue containing all tuples
    (v, v_i) with \{v, v_i\} \in E.
5 while queue is not empty do
        Remove front tuple (v_i, v_i) from queue.
        if L[i] = \infty then
             Add (v_i, v_k) to queue for all
              \{v_i,v_k\}\in E,\ k\neq i.
             Set L[i] \leftarrow L[i] + 1, pred[i] = i.
```



queue: 
$$(9,10)$$
,  $(9,6)$ ,  $(10,11)$ ,  $(10,9)_9$   
 $(11,10)$ ,  $(11,12)$ ,  $(7,3)$ ,  $(3,7)$ 

### Algorithm: BFS

```
: Graph G = (V, E), vertex v \in V.
  Input
              : d(v, y) for all y \in V and "a way of
  Output
                finding shortest paths".
 Number the vertices of G as v = v_1, \dots, v_n.
2 Let L[i] \leftarrow \infty for all i \in [n].
3 Let L[1] \leftarrow 0, pred[1] \leftarrow None.
4 Let queue be a queue containing all tuples
    (v, v_i) with \{v, v_i\} \in E.
5 while queue is not empty do
        Remove front tuple (v_i, v_i) from queue.
        if L[i] = \infty then
             Add (v_i, v_k) to queue for all
              \{v_i, v_k\} \in E, k \neq i.
             Set L[i] \leftarrow L[i] + 1, pred[i] = i.
```



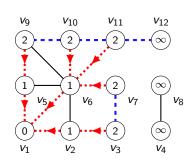
queue: 
$$(9,6)$$
,  $(10,11)$ ,  $(10,9)$ ,  $(11,10_9)$ ,  $(11,12)$ ,  $(7,3)$ ,  $(3,7)$ 

### Algorithm: BFS

: Graph G = (V, E), vertex  $v \in V$ . Input : d(v, y) for all  $y \in V$  and "a way of Output finding shortest paths". 1 Number the vertices of G as  $v = v_1, \dots, v_n$ .

- 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ .
- 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ .
- 4 Let queue be a queue containing all tuples  $(v, v_i)$  with  $\{v, v_i\} \in E$ .
- 5 while queue is not empty do

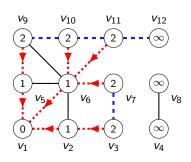
Remove front tuple  $(v_i, v_i)$  from queue. if  $L[i] = \infty$  then Add  $(v_i, v_k)$  to queue for all  $\{v_i,v_k\}\in E,\ k\neq i.$ Set  $L[i] \leftarrow L[i] + 1$ , pred[i] = i.



### Algorithm: BFS

: Graph G = (V, E), vertex  $v \in V$ . Input **Output** : d(v, y) for all  $y \in V$  and "a way of finding shortest paths". 1 Number the vertices of G as  $v = v_1, \dots, v_n$ . 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ . 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ . 4 Let queue be a queue containing all tuples  $(v, v_i)$  with  $\{v, v_i\} \in E$ . 5 while queue is not empty do Remove front tuple  $(v_i, v_i)$  from queue.

if  $L[i] = \infty$  then Add  $(v_i, v_k)$  to queue for all  $\{v_i, v_k\} \in E, k \neq i.$ Set  $L[i] \leftarrow L[i] + 1$ , pred[i] = i.

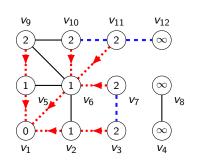


queue: 
$$(10,9)$$
,  $(11,10)$ ,  $(11,12)$ ,  $(7,3_9)$ ,  $(3,7)$ 

### Algorithm: BFS

: Graph G = (V, E), vertex  $v \in V$ . Input : d(v, y) for all  $y \in V$  and "a way of Output finding shortest paths". 1 Number the vertices of G as  $v = v_1, \dots, v_n$ . 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ . 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ . 4 Let queue be a queue containing all tuples  $(v, v_i)$  with  $\{v, v_i\} \in E$ . 5 while queue is not empty do

Remove front tuple  $(v_i, v_i)$  from queue.



queue:  $(11, 10), (11, 12), (7, 3), (3, 7)_9$ 

### Algorithm: BFS

```
Input : Graph G = (V, E), vertex v \in V.

Output : d(v, y) for all y \in V and "a way of finding shortest paths".

1 Number the vertices of G as v = v_1, \ldots, v_n.

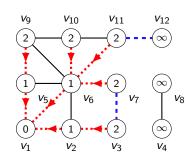
2 Let L[i] \leftarrow \infty for all i \in [n].

3 Let L[i] \leftarrow 0, pred[i] \leftarrow None.

4 Let queue be a queue containing all tuples (v, v_j) with \{v, v_j\} \in E.

5 while queue is not empty do

6 | Remove front tuple (v_i, v_j) from queue.
```



queue: (11, 12), (7, 3), (3, 7)

### Algorithm: BFS

```
Input : Graph G = (V, E), vertex v \in V.

Output : d(v, y) for all y \in V and "a way of finding shortest paths".

1 Number the vertices of G as v = v_1, \ldots, v_n.

2 Let L[I] \leftarrow \infty for all i \in [n].

3 Let L[I] \leftarrow 0, pred[I] \leftarrow None.

4 Let queue be a queue containing all tuples (v, v_j) with \{v, v_j\} \in E.
```

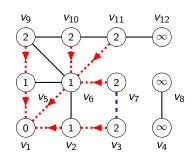
5 while queue is not empty do

Remove front tuple  $(v_i, v_j)$  from queue.

If  $L[j] = \infty$  then

Add  $(v_j, v_k)$  to queue for all  $\{v_j, v_k\} \in E, k \neq i$ .

 $\begin{cases} \{v_j, v_k\} \in E, \ k \neq i. \\ \text{Set L}[j] \leftarrow \text{L}[i] + 1, \ \text{pred}[j] = i. \end{cases}$ 



queue: (7,3),(3,7)

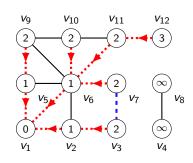
### Algorithm: BFS

Input : Graph G = (V, E), vertex  $v \in V$ . **Output** : d(v, y) for all  $y \in V$  and "a way of finding shortest paths". 1 Number the vertices of G as  $v = v_1, \dots, v_n$ . 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ . 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ . 4 Let queue be a queue containing all tuples  $(v, v_i)$  with  $\{v, v_i\} \in E$ .

5 while queue is not empty do

Remove front tuple  $(v_i, v_i)$  from queue. if  $L[i] = \infty$  then

Add  $(v_i, v_k)$  to queue for all  $\{v_j,v_k\}\in E,\ k\neq i.$ Set  $L[j] \leftarrow L[i] + 1$ , pred[j] = i. 9



queue: (7,3),(3,7)

#### Algorithm: BFS

```
Input : Graph G = (V, E), vertex v \in V.

Output : d(v, y) for all y \in V and "a way of finding shortest paths".

1 Number the vertices of G as v = v_1, \dots, v_n.

2 Let L[i] \leftarrow \infty for all i \in [n].

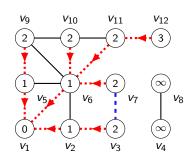
3 Let L[i] \leftarrow 0, pred[i] \leftarrow None.

4 Let queue be a queue containing all tuples (v, v_i) with \{v, v_i\} \in E.
```

5 while queue is not empty do

Remove front tuple  $(v_i, v_j)$  from queue. If  $L[j] = \infty$  then Add  $(v_i, v_k)$  to queue for all

 $\{v_j, v_k\} \in E, \ k \neq i.$ Set  $L[j] \leftarrow L[i] + 1$ , pred[j] = i.



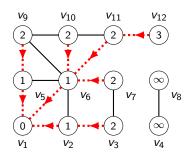
queue: (3,7)

Algorithm: BFS : Graph G = (V, E), vertex  $v \in V$ . Input **Output** : d(v, y) for all  $y \in V$  and "a way of finding shortest paths". 1 Number the vertices of G as  $v = v_1, \dots, v_n$ . 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ . 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ .

4 Let queue be a queue containing all tuples  $(v, v_i)$  with  $\{v, v_i\} \in E$ .

5 while queue is not empty do

Remove front tuple  $(v_i, v_i)$  from queue. if  $L[i] = \infty$  then Add  $(v_i, v_k)$  to queue for all  $\{v_j,v_k\}\in E,\ k\neq i.$ Set  $L[j] \leftarrow L[i] + 1$ , pred[j] = i. 9



#### Algorithm: BFS

```
Input : Graph G = (V, E), vertex v \in V.

Output : d(v, y) for all y \in V and "a way of finding shortest paths".

1 Number the vertices of G as v = v_1, \ldots, v_n.

2 Let L[i] \leftarrow \infty for all i \in [n].

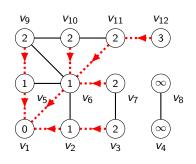
3 Let L[i] \leftarrow 0, pred[i] \leftarrow None.

4 Let queue be a queue containing all tuples (v, v_i) with \{v, v_i\} \in E.
```

5 while queue is not empty do

6 Remove front tuple  $(v_i, v_j)$  from queue. 7 if  $L[j] = \infty$  then 8 Add  $(v_j, v_k)$  to queue for all

 $\{v_j, v_k\} \in E, \ k \neq i.$   $\text{Set L[j]} \leftarrow \text{L[j]} + 1, \ \text{pred[j]} = i.$ 



#### Algorithm: BFS

```
Input : Graph G = (V, E), vertex v \in V.

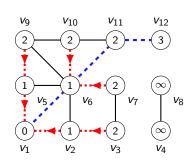
Output : d(v, y) for all y \in V and "a way of finding shortest paths".

1 Number the vertices of G as v = v_1, \ldots, v_n.
```

- 2 Let  $L[i] \leftarrow \infty$  for all  $i \in [n]$ .
- 3 Let  $L[1] \leftarrow 0$ , pred $[1] \leftarrow None$ .
- 4 Let queue be a queue containing all tuples  $(v, v_j)$  with  $\{v, v_j\} \in E$ .
- 5 while queue is not empty do
  6 | Remove front tuple (v<sub>i</sub>, v<sub>i</sub>) from queue.
- 7 if  $L[j] = \infty$  then 8  $Add(v_j, v_k)$  to queue for all  $\{v_j, v_k\} \in E, k \neq i$ . 9  $Set L[j] \leftarrow L[i] + 1$ , pred[j] = i.

10 Return L and pred.

In the output,  $L[i] = d(v, v_i)$ . By following edges back from  $v_i$  via pred, we can also quickly reconstruct a shortest path from v to  $v_i$ .



### Algorithm: BFS

```
: Graph G = (V, E), vertex v \in V.
  Output : d(v, y) for all y \in V and "a way of
                finding shortest paths".
1 Number the vertices of G as v = v_1, \dots, v_n.
2 Let L[i] \leftarrow \infty for all i \in [n].
3 Let L[1] \leftarrow 0, pred[1] \leftarrow None.
4 Let queue be a queue containing all tuples
```

 $(v, v_i)$  with  $\{v, v_i\} \in E$ . 5 while queue is not empty do

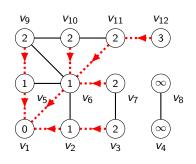
Remove front tuple  $(v_i, v_i)$  from queue. if  $L[j] = \infty$  then Add  $(v_i, v_k)$  to queue for all  $\{v_j, v_k\} \in E, k \neq i.$ Set  $L[j] \leftarrow L[i] + 1$ , pred[j] = i.

10 Return L and pred.

In the output,  $L[i] = d(v, v_i)$ . By following edges back from  $v_i$  via pred, we can also quickly reconstruct a shortest path from v to  $v_i$ .

9

E.g.  $v_1v_5v_{11}v_{12}$  is a shortest path from  $v_1$  to  $v_{12}$ .



#### Algorithm: BFS

```
Input : Graph G = (V, E), vertex v \in V.

Output : d(v, y) for all y \in V and "a way of finding shortest paths".

Number the vertices of G as v = v_1, \dots, v_n.

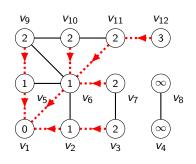
Let L[i] \leftarrow \infty for all i \in [n].

Let L[i] \leftarrow 0, pred[i] \leftarrow 0.
```

- 4 Let queue be a queue containing all tuples  $(v, v_j)$  with  $\{v, v_j\} \in E$ .
- 5 while queue is not empty do 6 Remove front tuple  $(v_i, v_j)$  from queue. 7 if  $L[j] = \infty$  then 8 Add  $(v_j, v_k)$  to queue for all
- $\begin{cases} \{v_j, v_k\} \in E, \ k \neq i. \\ \text{Set L}[j] \leftarrow \text{L}[i] + 1, \ \text{pred}[j] = i. \end{cases}$

10 Return L and pred.

**Time analysis:** If G is in adjacency list form, each edge is added to queue at most twice, incurring O(1) overhead each time, so the running time is O(|V| + |E|).



```
Algorithm: BFS
```

```
Input : Graph G = (V, E), vertex v \in V.

Output : d(v, y) for all y \in V and "a way of finding shortest paths".

1 Number the vertices of G as v = v_1, \dots, v_n.

2 Let L[I] \leftarrow \infty for all i \in [n].

3 Let L[I] \leftarrow 0, pred[I] \leftarrow None.

4 Let queue be a queue containing all tuples (v, v_j) with \{v, v_j\} \in E.
```

5 while queue is not empty do
6 Remove front tuple (v<sub>i</sub>, v<sub>i</sub>) from queue.

7 if 
$$L[j] = \infty$$
 then
8 Add  $(v_j, v_k)$  to queue for all  $\{v_j, v_k\} \in E, k \neq i$ .
9 Set  $L[j] \leftarrow L[i] + 1$ , pred $[j] = i$ .

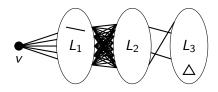
10 Return L and pred.

**Important:** There is a significant **space** inefficiency in this version of breadth-first search! See example sheet.

### BFS trees

**Definition:** A **BFS** tree T of G is a rooted tree (call its root x) with:

- **1** V(T) is the vertex set of a component of G;
- 2 The *i*'th layer of T is  $\{x: d_G(x, v) = i\}$ ;
- **③** If  $\{x,y\} \in E(G)$ , then  $|d_G(v,x) d_G(v,y)| \le 1$ , i.e. x and y must be in the same or adjacent layers of T.

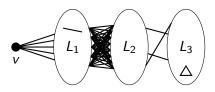


**Theorem:** The tree of edges from pred is always a BFS tree.

### BFS trees

**Definition:** A **BFS** tree T of G is a rooted tree (call its root x) with:

- **1** V(T) is the vertex set of a component of G;
- ② The *i*'th layer of T is  $\{x: d_G(x, v) = i\}$ ;
- **③** If  $\{x,y\} \in E(G)$ , then  $|d_G(v,x) d_G(v,y)| \le 1$ , i.e. x and y must be in the same or adjacent layers of T.



**Theorem:** The tree of edges from pred is always a BFS tree.

**Proof:** We already proved (1) and (2), so suppose  $\{x,y\} \in E(G)$ .

If P is a shortest path from v to x, then Pxy is a path from v to y, so  $d(v,y) \le d(v,x) + 1$ . Likewise  $d(v,x) \le d(v,y) + 1$ .

John Lapinskas Video 4-3 6/6