

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 ?

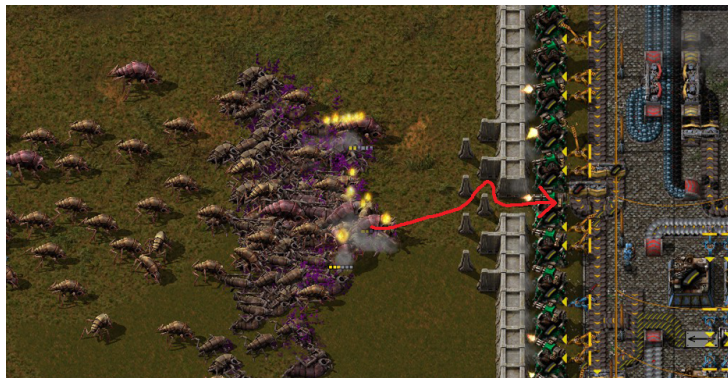
E.g. can an enemy attack the base without breaking down a wall?



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 ?

Graph distance

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

What do we mean by “shortest”?

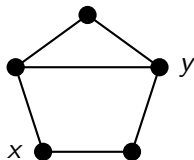
The **distance** between x and y , $d(x, y)$, is the length in edges of a shortest path between x and y , or ∞ if no such path exists.

Graph distance

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 ∞ if no such path exists.

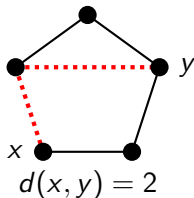


Graph distance

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 ∞ if no such path exists.

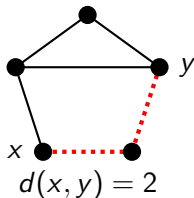


Graph distance

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 ∞ if no such path exists.

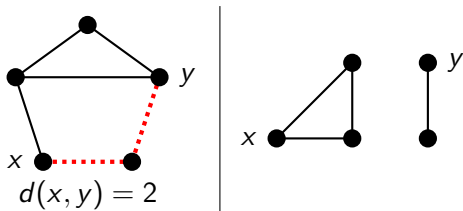


Graph distance

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 ∞ if no such path exists.

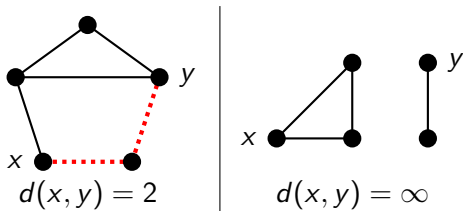


Graph distance

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 ∞ if no such path exists.

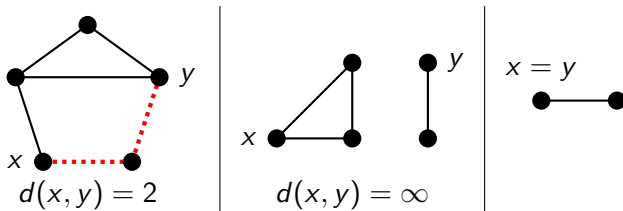


Graph distance

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 ∞ if no such path exists.

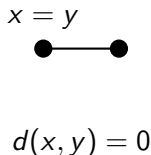
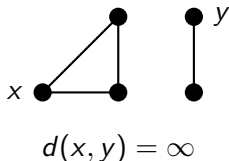
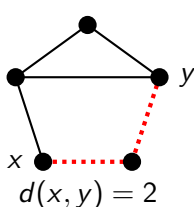


Graph distance

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 ∞ if no such path exists.

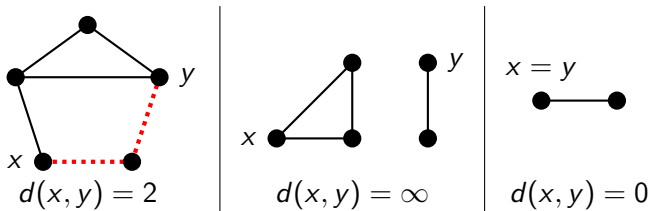


Graph distance

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 ∞ if no such path exists.

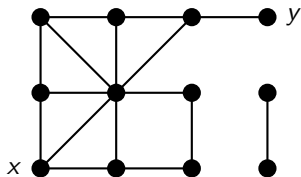


In directed graphs, it's the same except that the path is **from** x **to** y . So... we might not have $d(x, y) = d(y, x)$!

Breadth-first search: The idea

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

Output: A shortest path from x to y .

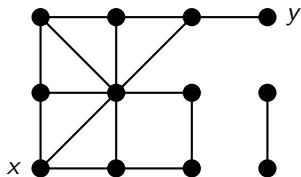


Let L_i be the set of vertices at distance i from x .

Breadth-first search: The idea

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

Output: A shortest path from x to y .

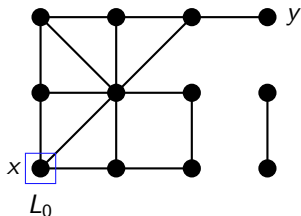


Let L_i be the set of vertices at distance i from x . So $L_0 = \{x\}$.

Breadth-first search: The idea

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

Output: A shortest path from x to y .

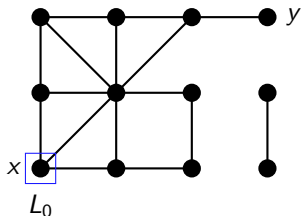


Let L_i be the set of vertices at distance i from x . So $L_0 = \{x\}$.

Breadth-first search: The idea

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

Output: A shortest path from x to y .



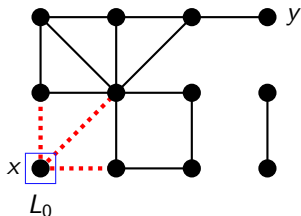
Let L_i be the set of vertices at distance i from x . So $L_0 = \{x\}$.

L_1 is everything adjacent to x .

Breadth-first search: The idea

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

Output: A shortest path from x to y .



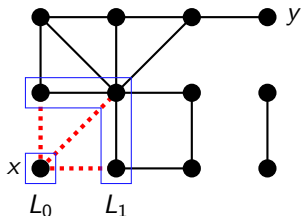
Let L_i be the set of vertices at distance i from x . So $L_0 = \{x\}$.

L_1 is everything adjacent to x .

Breadth-first search: The idea

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

Output: A shortest path from x to y .



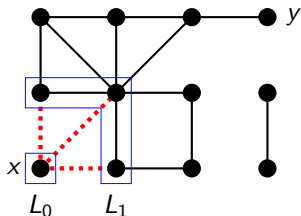
Let L_i be the set of vertices at distance i from x . So $L_0 = \{x\}$.

L_1 is everything adjacent to x .

Breadth-first search: The idea

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 $L_0 = \{x\}$.

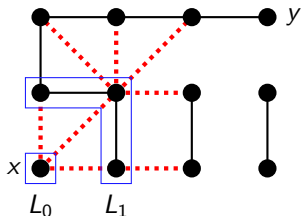
L_1 is everything adjacent to x .

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

Breadth-first search: The idea

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 $L_0 = \{x\}$.

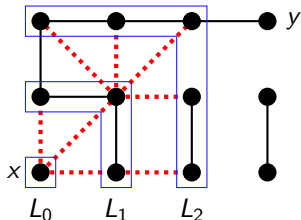
L_1 is everything adjacent to x .

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

Breadth-first search: The idea

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 $L_0 = \{x\}$.

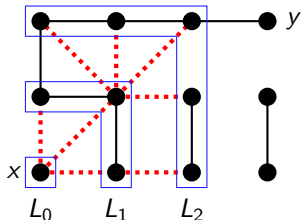
L_1 is everything adjacent to x .

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

Breadth-first search: The idea

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 .

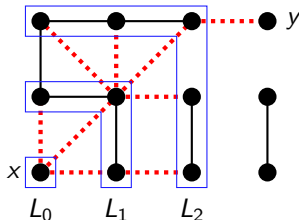
In general, L_{i+1} is everything adjacent to L_i and not in $L_0 \cup \dots \cup L_i$.

By continuing this until we find y , keeping track of which edges we use, we get a shortest path to y .

Breadth-first search: The idea

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 .

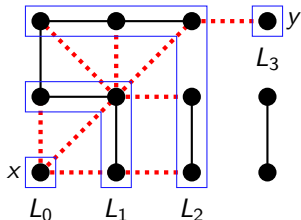
In general, L_{i+1} is everything adjacent to L_i and not in $L_0 \cup \dots \cup L_i$.

By continuing this until we find y , keeping track of which edges we use, we get a shortest path to y .

Breadth-first search: The idea

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 .

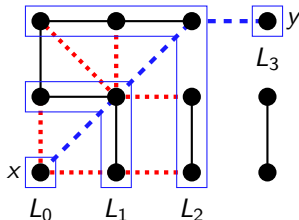
In general, L_{i+1} is everything adjacent to L_i and not in $L_0 \cup \dots \cup L_i$.

By continuing this until we find y , keeping track of which edges we use, we get a shortest path to y .

Breadth-first search: The idea

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 $L_0 = \{x\}$.

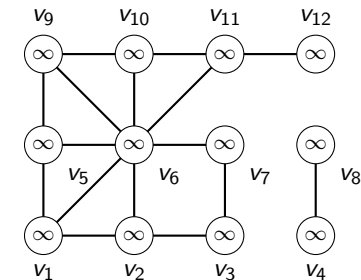
L_1 is everything adjacent to x .

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 $L_0 \cup \dots \cup L_i$.

By continuing this until we find y , keeping track of which edges we use, we get a shortest path to y .

Breadth-first search: Implementation



queue:

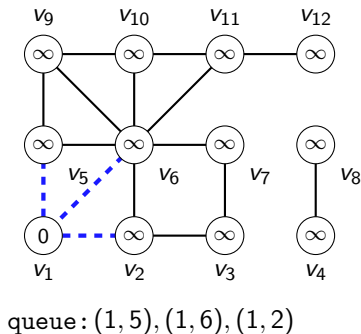
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E, k \neq j$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 Return L and pred .
-

Breadth-first search: Implementation



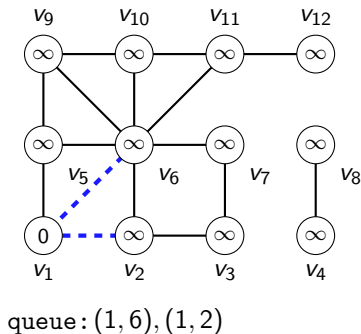
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq j$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



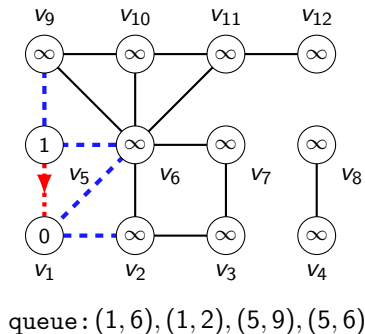
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq j$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 Return L and pred .
-

Breadth-first search: Implementation



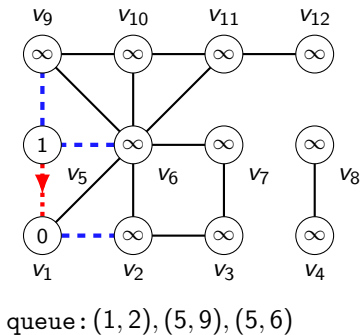
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq j$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



Algorithm: BFS

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

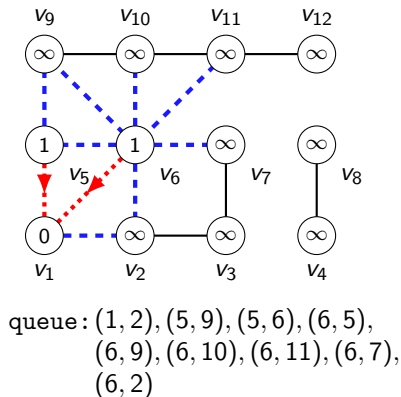
Output : $d(v, y)$ for all $y \in V$ and “a way of finding shortest paths”.

- ```

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
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
8 Add (v_j, v_k) to queue for all
 $\{v_j, v_k\} \in E$, $k \neq i$.
9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
10 Return L and pred .

```

# Breadth-first search: Implementation



---

## Algorithm: BFS

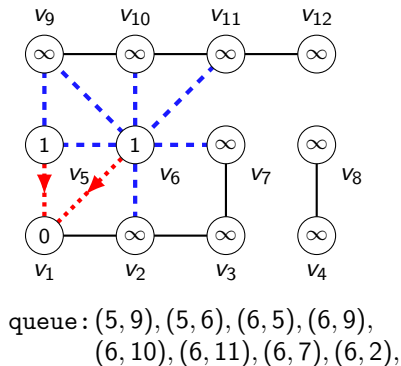
---

**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  $L[1] \leftarrow 0$ ,  $\text{pred}[1] \leftarrow \text{None}$ .
  - 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**
      - 8         Add  $(v_j, v_k)$  to queue for all  $\{v_j, v_k\} \in E, k \neq j$ .
      - 9         Set  $L[j] \leftarrow L[i] + 1$ ,  $\text{pred}[j] = i$ .
  - 10 **Return**  $L$  and  $\text{pred}$ .
-

# Breadth-first search: Implementation



---

## Algorithm: BFS

---

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

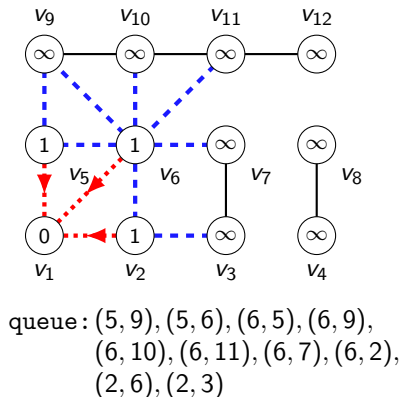
**Output** :  $d(v, y)$  for all  $y \in V$  and "a way of finding shortest paths".

---

- 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  $L[1] \leftarrow 0$ ,  $\text{pred}[1] \leftarrow \text{None}$ .
  - 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**
      - 8         Add  $(v_j, v_k)$  to queue for all  $\{v_j, v_k\} \in E, k \neq j$ .
      - 9         Set  $L[j] \leftarrow L[i] + 1$ ,  $\text{pred}[j] = i$ .
  - 10 **Return**  $L$  and  $\text{pred}$ .
-



# Breadth-first search: Implementation



---

## Algorithm: BFS

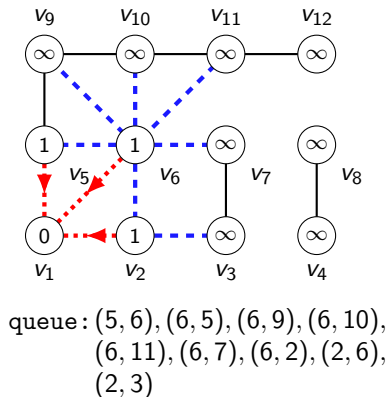
---

**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  $L[1] \leftarrow 0$ ,  $\text{pred}[1] \leftarrow \text{None}$ .
  - 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**
      - 8         Add  $(v_j, v_k)$  to queue for all  $\{v_j, v_k\} \in E$ ,  $k \neq j$ .
      - 9         Set  $L[j] \leftarrow L[i] + 1$ ,  $\text{pred}[j] = i$ .
  - 10 **Return**  $L$  and  $\text{pred}$ .
-

# Breadth-first search: Implementation



---

## Algorithm: BFS

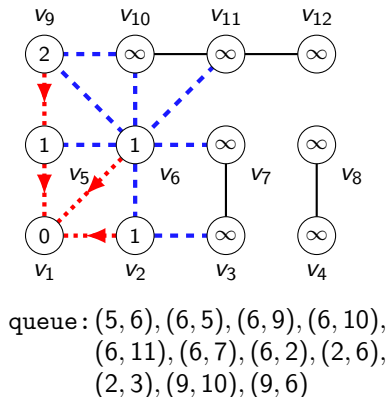
---

**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  $L[1] \leftarrow 0$ ,  $\text{pred}[1] \leftarrow \text{None}$ .
  - 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**
      - 8         Add  $(v_j, v_k)$  to queue for all  $\{v_j, v_k\} \in E$ ,  $k \neq j$ .
      - 9         Set  $L[j] \leftarrow L[i] + 1$ ,  $\text{pred}[j] = i$ .
  - 10 **Return**  $L$  and  $\text{pred}$ .
-

# Breadth-first search: Implementation




---

## Algorithm: BFS

---

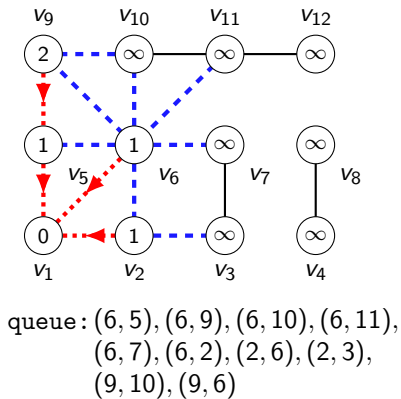
**Input** : Graph  $G = (V, E)$ , vertex  $v \in V$ .

**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  $L[1] \leftarrow 0$ ,  $\text{pred}[1] \leftarrow \text{None}$ .
  - 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**
      - 8         Add  $(v_j, v_k)$  to queue for all  $\{v_j, v_k\} \in E$ ,  $k \neq i$ .
      - 9         Set  $L[j] \leftarrow L[i] + 1$ ,  $\text{pred}[j] = i$ .
  - 10 **Return**  $L$  and  $\text{pred}$ .
-

## Breadth-first search: Implementation



### Algorithm: BFS

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

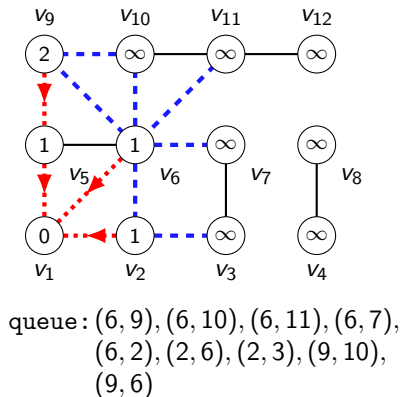
**Output** :  $d(v, y)$  for all  $y \in V$  and “a way of finding shortest paths”.

- ```

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  $L[1] \leftarrow 0$ ,  $\text{pred}[1] \leftarrow \text{None}$ .
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
8          Add  $(v_j, v_k)$  to queue for all
            $\{v_j, v_k\} \in E$ ,  $k \neq i$ .
9          Set  $L[j] \leftarrow L[i] + 1$ ,  $\text{pred}[j] = i$ .
10 Return  $L$  and  $\text{pred}$ .

```

Breadth-first search: Implementation



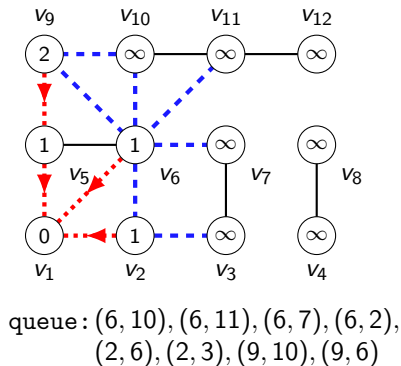
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq i$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



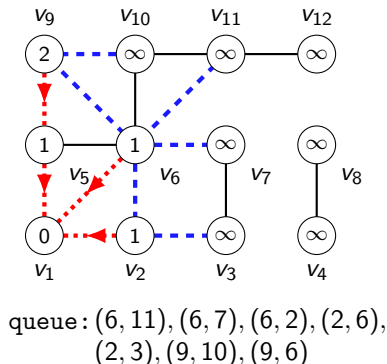
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E, k \neq j$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



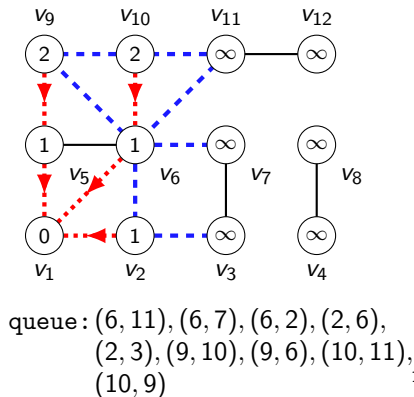
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E, k \neq j$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



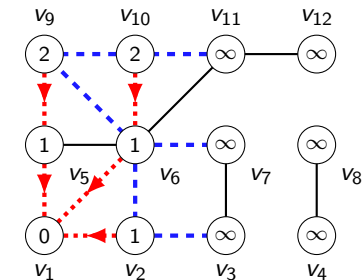
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq i$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



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

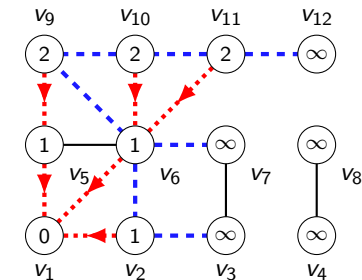
Algorithm: BFS

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 $v = v_1, \dots, v_n$.
 - 2 Let $L[i] \leftarrow \infty$ for all $i \in [n]$.
 - 3 Let $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq j$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



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

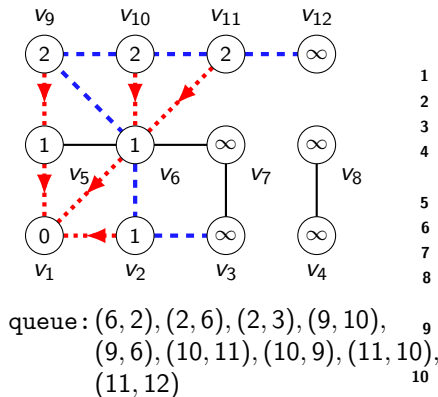
Algorithm: BFS

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".

- 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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq i$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



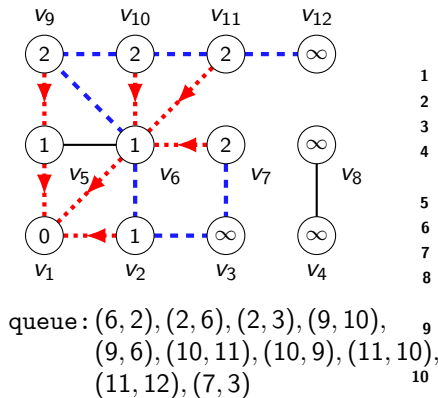
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq j$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



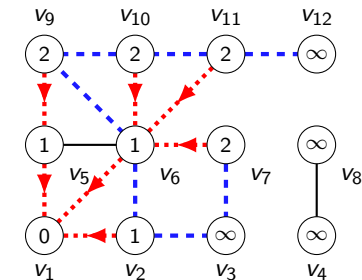
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq i$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



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

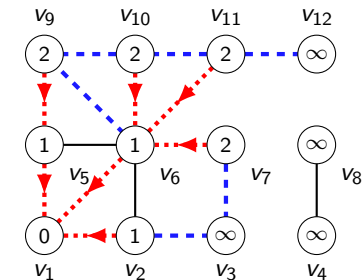
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq i$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



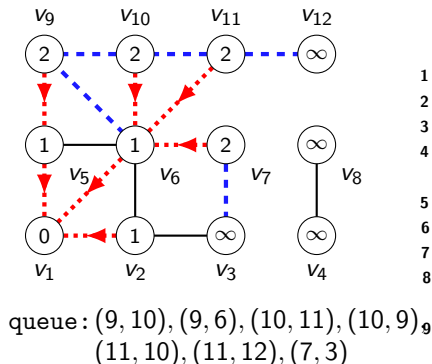
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E, k \neq i$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



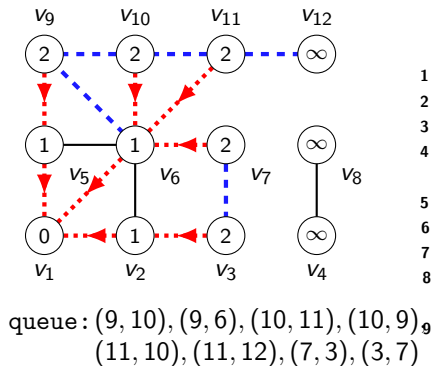
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
- 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E, k \neq j$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
- 10 **Return** L and pred .

Breadth-first search: Implementation



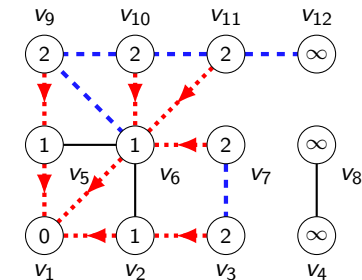
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq j$.
 - Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



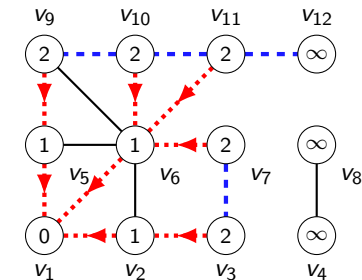
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq i$.
 - Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



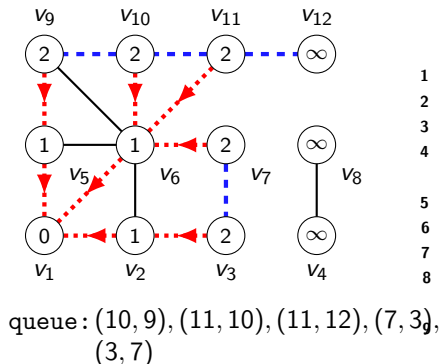
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq i$.
 - Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 Return L and pred .
-

Breadth-first search: Implementation



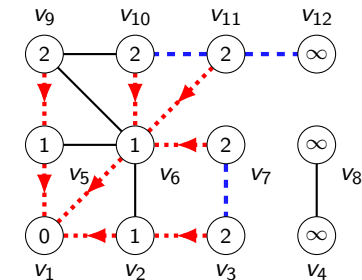
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E, k \neq i$.
 - Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 Return L and pred .
-

Breadth-first search: Implementation



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

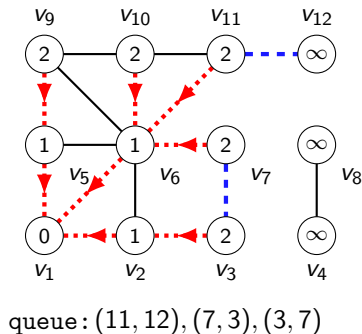
Algorithm: BFS

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 $v = v_1, \dots, v_n$.
 - 2 Let $L[i] \leftarrow \infty$ for all $i \in [n]$.
 - 3 Let $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E, k \neq i$.
 - Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



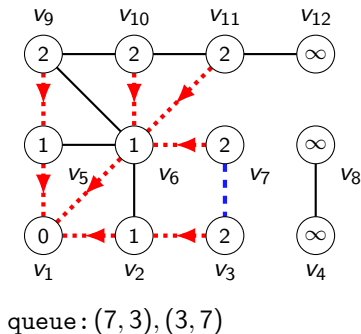
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq i$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 **Return** L and pred .
-

Breadth-first search: Implementation



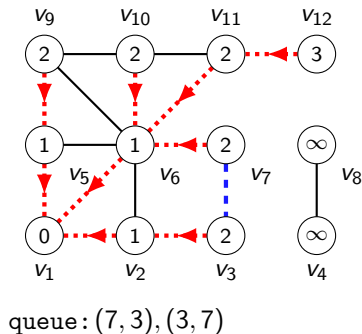
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E, k \neq i$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 Return L and pred .
-

Breadth-first search: Implementation



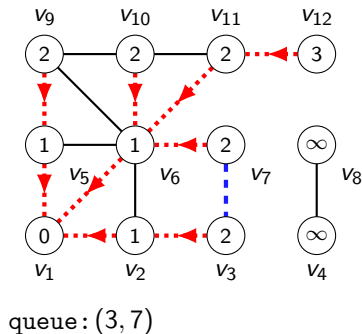
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq i$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 Return L and pred .
-

Breadth-first search: Implementation



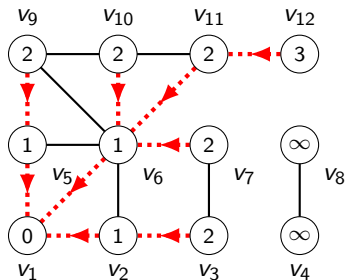
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E$, $k \neq i$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 Return L and pred .
-

Breadth-first search: Implementation



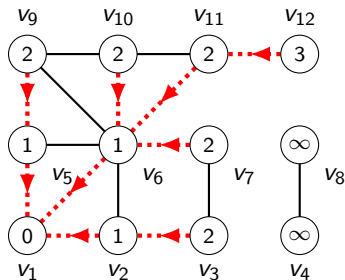
Algorithm: BFS

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E, k \neq j$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 Return L and pred .
-

Breadth-first search: Implementation



Algorithm: BFS

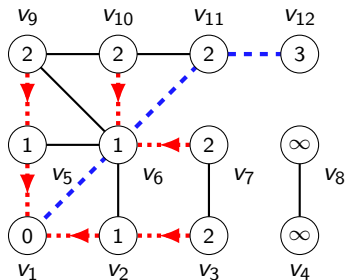
Input : Graph $G = (V, E)$, vertex $v \in V$.

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E, k \neq j$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 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 .

Breadth-first search: Implementation



Algorithm: BFS

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

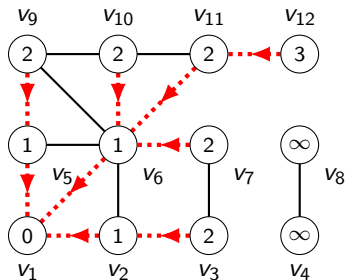
Output : $d(v, y)$ for all $y \in V$ and "a way of finding shortest paths".

- 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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E, k \neq i$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 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 .

E.g. $v_1 v_6 v_{11} v_{12}$ is a shortest path from v_1 to v_{12} .

Breadth-first search: Implementation



Algorithm: BFS

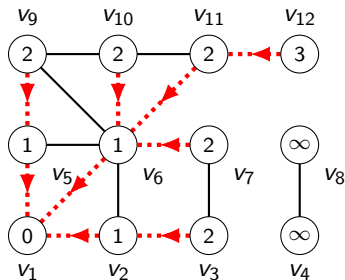
Input : Graph $G = (V, E)$, vertex $v \in V$.

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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E, k \neq i$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 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 $O(|V| + |E|)$.

Breadth-first search: Implementation



Algorithm: BFS

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

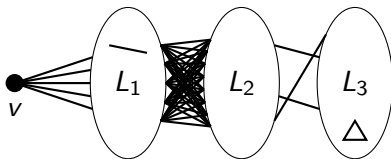
Output : $d(v, y)$ for all $y \in V$ and "a way of finding shortest paths".

- 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 $L[1] \leftarrow 0$, $\text{pred}[1] \leftarrow \text{None}$.
 - 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**
 - 8 Add (v_j, v_k) to queue for all $\{v_j, v_k\} \in E, k \neq i$.
 - 9 Set $L[j] \leftarrow L[i] + 1$, $\text{pred}[j] = i$.
 - 10 Return L and pred .
-

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

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

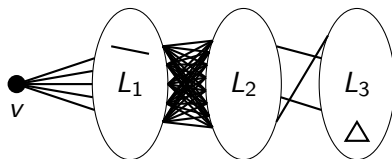
- 1 $V(T)$ is the vertex set of a component of G ;
- 2 The i 'th layer of T is $\{x: d_G(x, v) = i\}$;
- 3 If $\{x, y\} \in E(G)$, then $|d_G(v, x) - d_G(v, y)| \leq 1$, i.e. x and y must be in the same or adjacent layers of T .



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

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 $|d_G(v, x) - d_G(v, y)| \leq 1$, 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 P_{xy} is a path from v to y , so $d(v, y) \leq d(v, x) + 1$. Likewise $d(v, x) \leq d(v, y) + 1$. ✓