Algorithms and Data Structures (II)

Gabriel Istrate

May 27, 2024

Where are we:

Why did we want dynamic sets to start with ?

- Graph algorithms.
- Goal: To see some algorithms that use data structures.

We live in a highly connected world …

... and that's important.

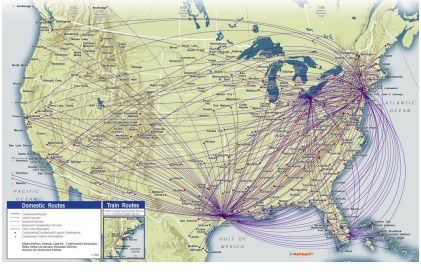
A certain disease from Wuhan, China had dramatic effects all over the planet ...

A software bug in the alarm system at the control room of FirstEnergy, in Akron, Ohio knocks out the power grid in the whole Northeast United States (2003).

How do real networks look like ?

How do network properties impact **the processes** that take place on them ?

Some real networks



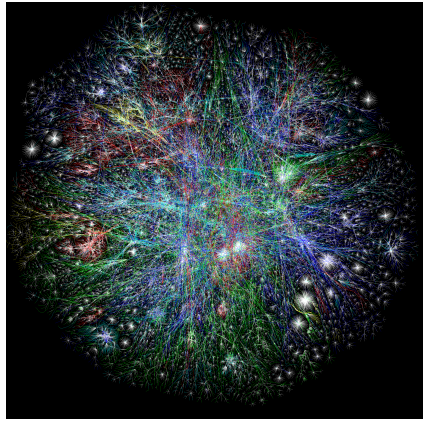
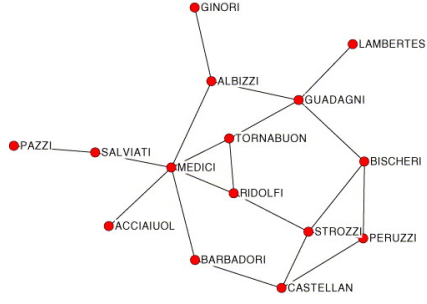
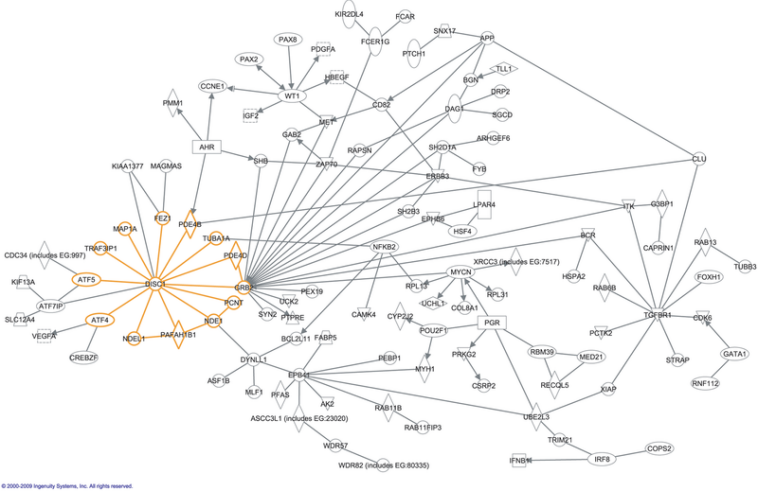


Figure: (a). Air traffic map of the U.S. (b). Physical Internet

Marriage Networks of important families in Medieval Florence.

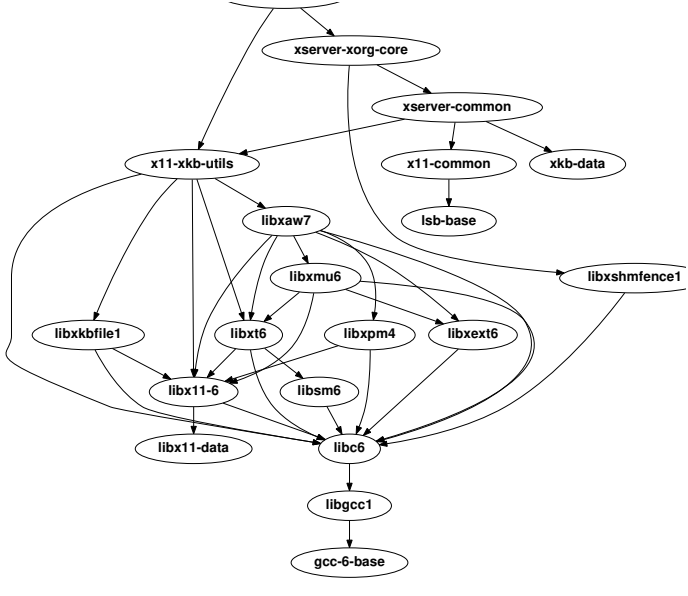


Interactome ...

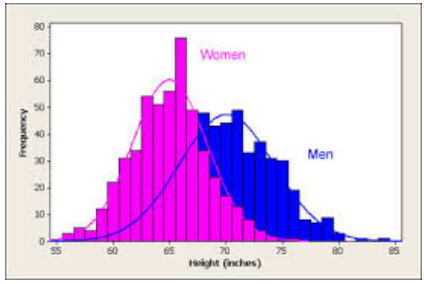


Part of the DISC1 interactome, with genes represented by text in boxes and interactions noted by lines between the genes. From Hennah and Porteous (2009).

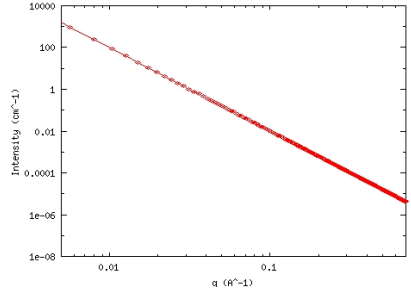
… or even …


xserver-xorg-core
xserver-common
x11-xkb-utils
x11-common
xkb-data
lsb-base
libxaw7
libxmu6
libxshmfence1
libxkbfile1
libxt6
libxpm4
libxext6
libx11-6
libsm6
libx11-data
libc6
libgcc1
gcc-6-base

What's so interesting about networks ?



Small worlds: everyone is "not very far from everyone".



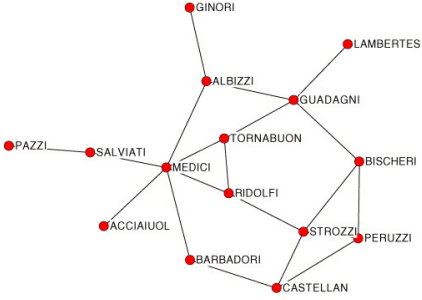
(a). Distribution of heights in the U.S. population.

(b). Degree distribution (aproximately) power law. Few "tall" people, "many" well connected people.

What's so interesting about networks ? (II)

Centrality

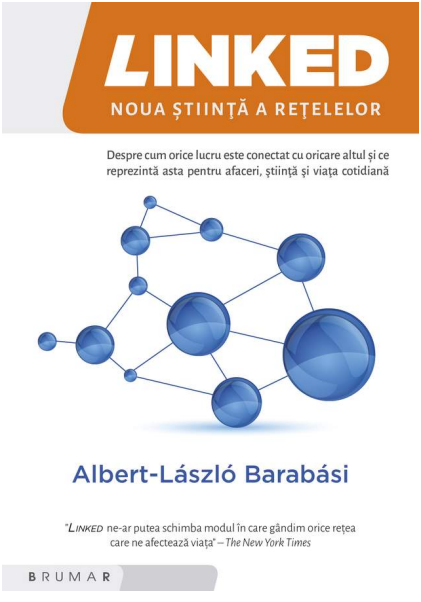
Some nodes are "more important/central than others".



How do we measure centrality ?

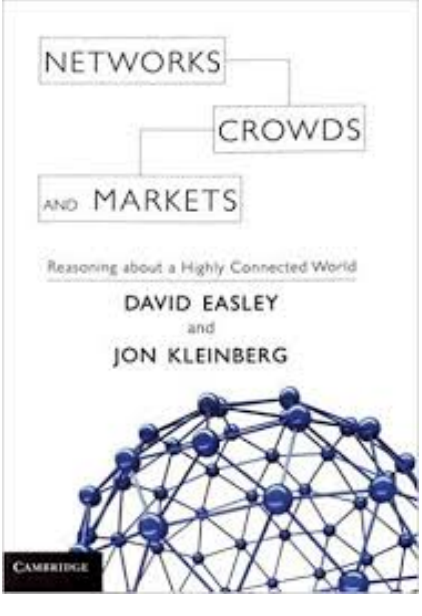
- degree centrality: $c(y) = \frac{1}{n-1} deg(y)$.
- betweenness centrality: $c(y) = \sum_{x \neq z} \frac{\sigma_{x,z}[y]}{\sigma_{x,z}}$
- eigenvector centrality: $c(y) = w(y)$, w the eigenvector of the adjacency matrix A of G corresponding to the largest eigenvalue.

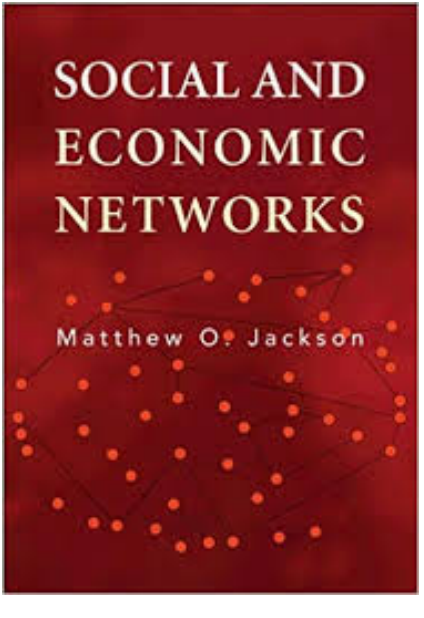
Want to read something interesting ?



By the way, not only in America ...

Want to read something even more interesting ?





Can you study large networks without good algorithms ?

To conclude: Many Models and Applications

- Social networks: *who knows who*
- The Web graph: *which page links to which*
- The Internet graph: *which router links to which*
- Citation graphs: *who references whose papers*
- Planar graphs: *which country is next to which*
- Well-shaped meshes: *pretty pictures with triangles*
- Geometric graphs: *who is near who*

Definitions

- A ***graph***

$$G = (V, E)$$

- V is the set of ***vertices*** (also called ***nodes***)
- E is the set of ***edges***

Definitions

- A ***graph***

$$G = (V, E)$$

- V is the set of ***vertices*** (also called ***nodes***)
- E is the set of ***edges***
  - $E \subseteq V \times V$, i.e., E is a ***relation between vertices***
  - an edge $e = (u, v) \in V$ is a pair of vertices $u \in V$ and $v \in V$

Definitions

- A ***graph***

$$G = (V, E)$$

- V is the set of ***vertices*** (also called ***nodes***)
- E is the set of ***edges***
  - $E \subseteq V \times V$, i.e., E is a ***relation between vertices***
  - an edge $e = (u, v) \in V$ is a pair of vertices $u \in V$ and $v \in V$
- An *undirected* graph is characterized by a *symmetric* relation between vertices
  - an edge is a set $e = \{u, v\}$ of two vertices

Graph Representation

- How do we represent a graph $G = (E, V)$ in a computer?

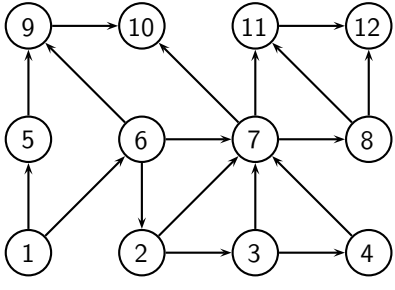
Graph Representation

- How do we represent a graph $G = (E, V)$ in a computer?
- *Adjacency-list representation*
- $V = \{1, 2, \ldots |V|\}$
- G consists of an array *Adj*
- A vertex $u \in V$ is represented by an element in the array *Adj*

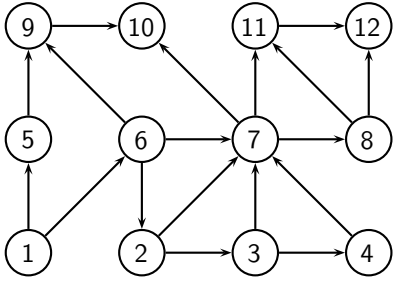
Graph Representation

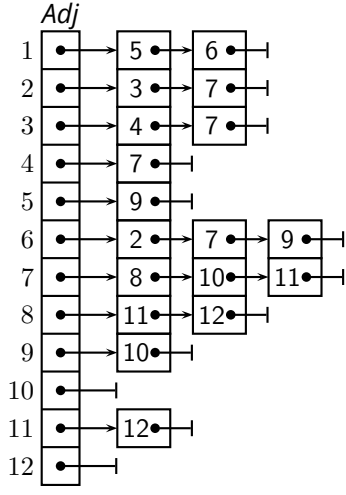
- How do we represent a graph $G = (E, V)$ in a computer?
- *Adjacency-list representation*
- $V = \{1, 2, \ldots |V|\}$
- G consists of an array Adj
- A vertex $u \in V$ is represented by an element in the array Adj
- $Adj[u]$ is the ***adjacency list*** of vertex u
  - the list of the vertices that are adjacent to u
  - i.e., the list of all v such that $(u, v) \in E$

Example

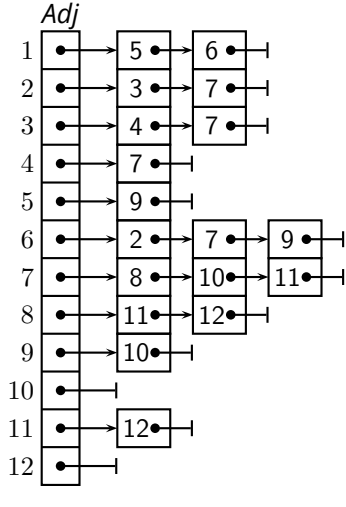


Example


9
10
11
12
5
6
7
8
1
2
3
4

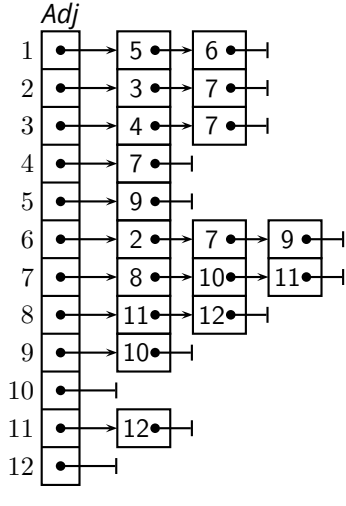

Adj
1
2
3
4
5
6
7
8
9
10
11
12
5
6
3
7
4
7
7
9
2
7
9
8
10
11
11
12
10
12

Using the Adjacency List



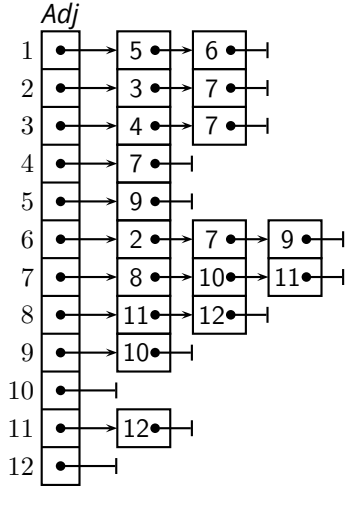
Using the Adjacency List

- Accessing a vertex u? ✓.



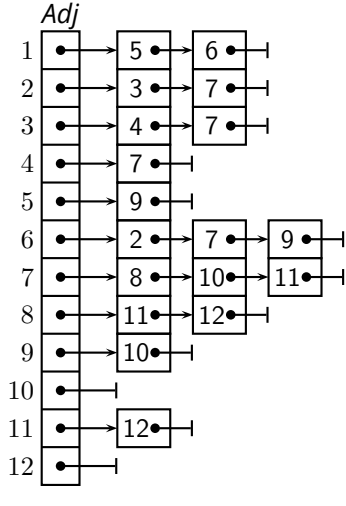
Using the Adjacency List

- Accessing a vertex u? $O(1)$ ✓.
  - optimal ✓



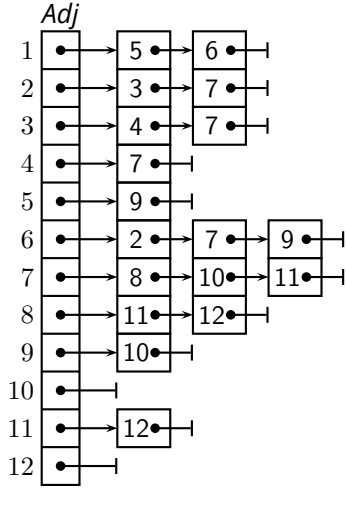
Using the Adjacency List

- Accessing a vertex u? $O(1)$ ✓.
  - optimal ✓
- Iteration through V?



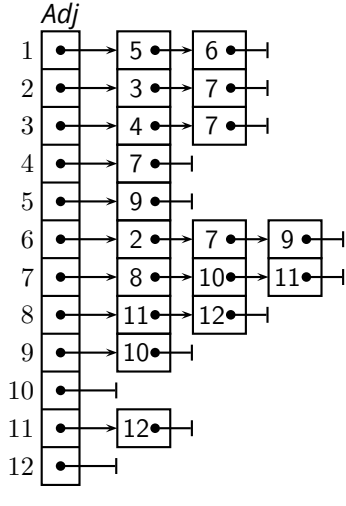
Using the Adjacency List

- Accessing a vertex u? $O(1)$ ✓.
  - optimal ✓
- Iteration through V? $\Theta(|V|)$
  - optimal ✓



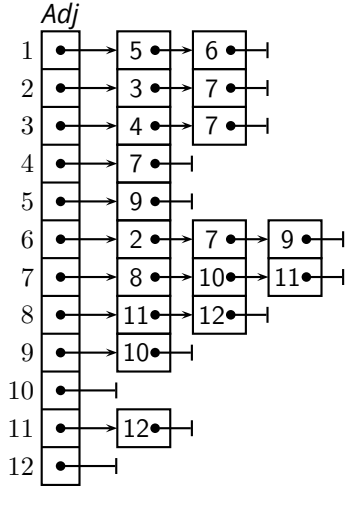
Using the Adjacency List

- Accessing a vertex u? $O(1)$ ✓.
  - optimal ✓
- Iteration through V? $\Theta(|V|)$
  - optimal ✓
- Iteration through E?



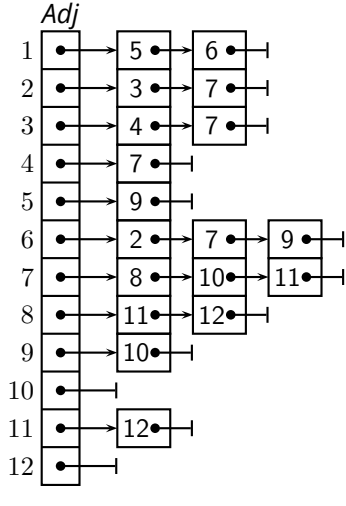
Using the Adjacency List

- Accessing a vertex u? $O(1)$ ✓.
  - optimal ✓
- Iteration through V? $\Theta(|V|)$
  - optimal ✓
- Iteration through E? $\Theta(|V| + |E|)$
  - okay (not optimal)



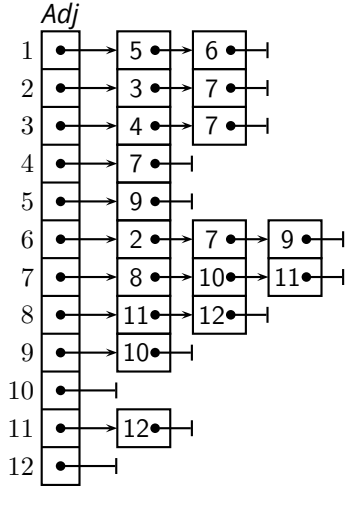
Using the Adjacency List

- Accessing a vertex u? $O(1)$ ✓.
  - optimal ✓
- Iteration through V? $\Theta(|V|)$
  - optimal ✓
- Iteration through E? $\Theta(|V| + |E|)$
  - okay (not optimal)
- Checking $(u, v) \in E$?



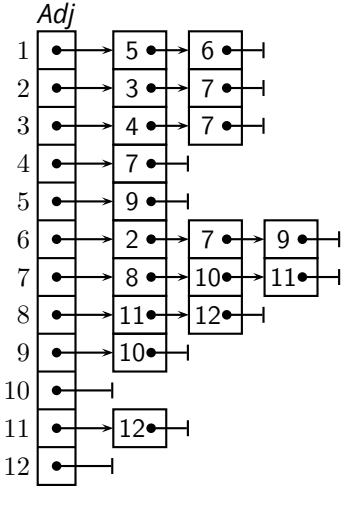
Using the Adjacency List

- Accessing a vertex u? $O(1)$ ✓.
  - optimal ✓
- Iteration through V? $\Theta(|V|)$
  - optimal ✓
- Iteration through E? $\Theta(|V| + |E|)$
  - okay (not optimal)
- Checking $(u, v) \in E$? $O(|V|)$



Using the Adjacency List

- Accessing a vertex u? $O(1)$ ✓.
  - optimal ✓
- Iteration through V? $\Theta(|V|)$
  - optimal ✓
- Iteration through E? $\Theta(|V| + |E|)$
  - okay (not optimal)
- Checking $(u, v) \in E$? $O(|V|)$
  - bad ×



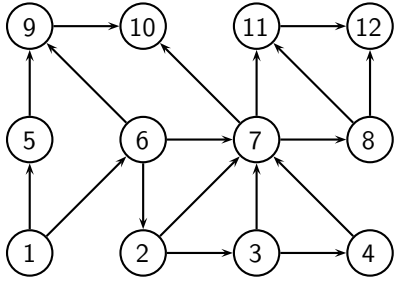
Graph Representation (2)

Graph Representation (2)

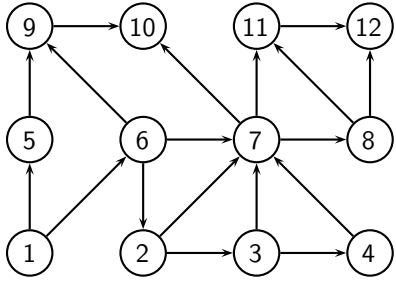
- *Adjacency-matrix representation*
- $V = \{1, 2, \ldots |V|\}$
- G consists of a $|V| \times |V|$ matrix A
- $A = (a_{ij})$ such that

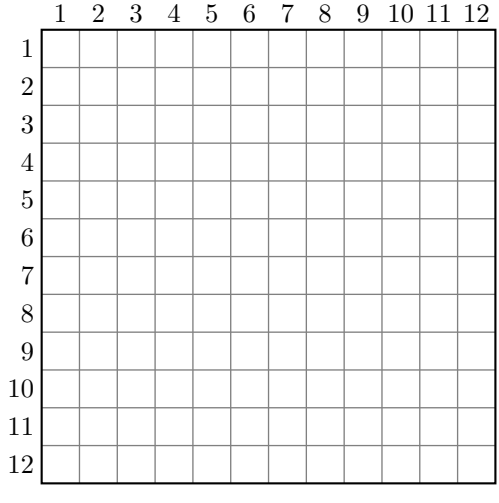
$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Example

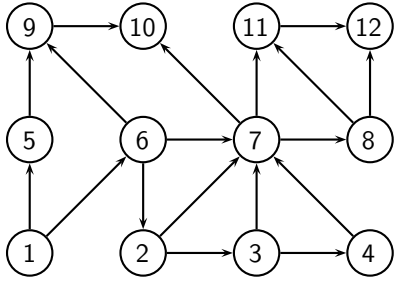


Example



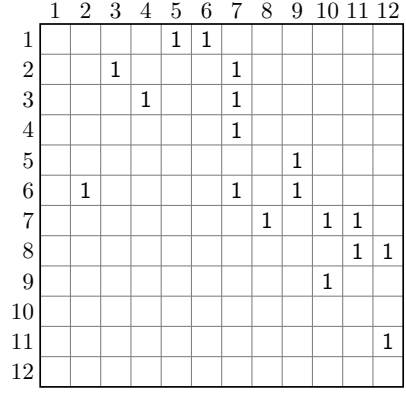


Example



	1	2	3	4	5	6	7	8	9	10	11	12
1					1	1						
2			1				1					
3				1			1					
4							1					
5									1			
6		1					1		1			
7								1		1	1	
8											1	1
9										1		
10												
11												1
12												

Using the Adjacency Matrix



	1	2	3	4	5	6	7	8	9	10	11	12
1					1	1						
2			1				1					
3				1			1					
4							1					
5									1			
6		1					1		1			
7								1		1	1	
8											1	1
9										1		
10												
11												1
12												

Using the Adjacency Matrix

- Accessing a vertex u?

	1	2	3	4	5	6	7	8	9	10	11	12
1					1	1						
2			1				1					
3				1			1					
4							1					
5									1			
6		1					1		1			
7								1		1	1	
8											1	1
9										1		
10												
11												1
12												

Using the Adjacency Matrix

- Accessing a vertex u? $O(1)$
  - optimal ✓

	1	2	3	4	5	6	7	8	9	10	11	12
1					1	1						
2			1				1					
3				1			1					
4							1					
5									1			
6		1					1		1			
7								1		1	1	
8											1	1
9										1		
10												
11												1
12												

Using the Adjacency Matrix

- Accessing a vertex u? $O(1)$
  - optimal ✓
- Iteration through V?

	1	2	3	4	5	6	7	8	9	10	11	12
1					1	1						
2			1				1					
3				1			1					
4							1					
5									1			
6		1					1		1			
7								1		1	1	
8											1	1
9										1		
10												
11												1
12												

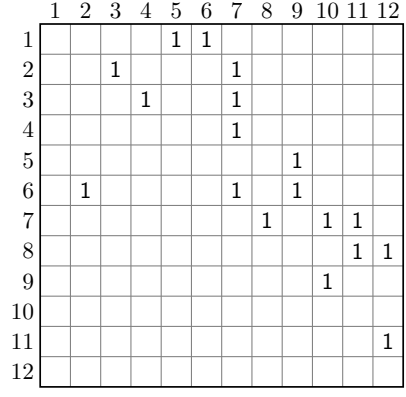
Using the Adjacency Matrix

- Accessing a vertex u? $O(1)$
  - optimal ✓
- Iteration through V? $\Theta(|V|)$
  - optimal ✓

	1	2	3	4	5	6	7	8	9	10	11	12
1					1	1						
2			1				1					
3				1			1					
4							1					
5									1			
6		1					1		1			
7								1		1	1	
8											1	1
9										1		
10												
11												1
12												

Using the Adjacency Matrix

- Accessing a vertex u? $O(1)$
  - optimal ✓
- Iteration through V? $\Theta(|V|)$
  - optimal ✓
- Iteration through E?



	1	2	3	4	5	6	7	8	9	10	11	12
1					1	1						
2			1				1					
3				1			1					
4							1					
5									1			
6		1					1		1			
7								1		1	1	
8											1	1
9										1		
10												
11												1
12												

Using the Adjacency Matrix

- Accessing a vertex u? $O(1)$
  - optimal ✓
- Iteration through V? $\Theta(|V|)$
  - optimal ✓
- Iteration through E? $\Theta(|V|^2)$
  - possibly very bad ×.

	1	2	3	4	5	6	7	8	9	10	11	12
1					1	1						
2			1				1					
3				1			1					
4							1					
5									1			
6		1					1		1			
7								1		1	1	
8											1	1
9										1		
10												
11												1
12												

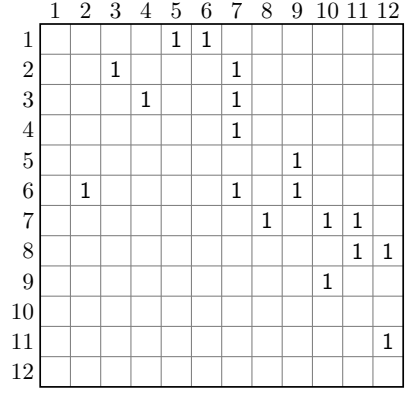
Using the Adjacency Matrix

- Accessing a vertex u? $O(1)$
  - optimal ✓
- Iteration through V? $\Theta(|V|)$
  - optimal ✓
- Iteration through E? $\Theta(|V|^2)$
  - possibly very bad ×.
- Checking $(u, v) \in E$?

	1	2	3	4	5	6	7	8	9	10	11	12
1					1	1						
2			1				1					
3				1			1					
4							1					
5									1			
6		1					1		1			
7								1		1	1	
8											1	1
9										1		
10												
11												1
12												

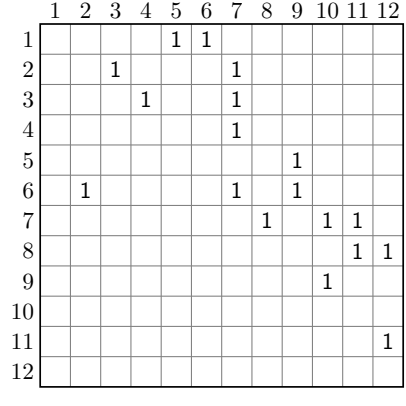
Using the Adjacency Matrix

- Accessing a vertex u? $O(1)$
  - optimal ✓
- Iteration through V? $\Theta(|V|)$
  - optimal ✓
- Iteration through E? $\Theta(|V|^2)$
  - possibly very bad ×.
- Checking $(u, v) \in E$? $O(1)$



Using the Adjacency Matrix

- Accessing a vertex u? $O(1)$
  - optimal ✓
- Iteration through V? $\Theta(|V|)$
  - optimal ✓
- Iteration through E? $\Theta(|V|^2)$
  - possibly very bad ×.
- Checking $(u, v) \in E$? $O(1)$
  - optimal ✓



Space Complexity

Space Complexity

- Adjacency-list representation

Space Complexity

- Adjacency-list representation

$$\Theta(|V| + |E|)$$

Space Complexity

- Adjacency-list representation

$$\Theta(|V| + |E|)$$

optimal .

Space Complexity

- Adjacency-list representation

$$\boxed{\Theta(|V| + |E|)}$$

  optimal .

- Adjacency-matrix representation

Space Complexity

- Adjacency-list representation

$$\Theta(|V| + |E|)$$

  optimal .

- Adjacency-matrix representation

$$\Theta(|V|^2)$$

Space Complexity

- Adjacency-list representation

$$\Theta(|V| + |E|)$$

optimal .

- Adjacency-matrix representation

$$\Theta(|V|^2)$$

possibly very bad ×.

Space Complexity

- Adjacency-list representation

$$\Theta(|V| + |E|)$$

  optimal .

- Adjacency-matrix representation

$$\Theta(|V|^2)$$

  possibly very bad ×.

- When is the adjacency-matrix “very bad”?

Choosing a Graph Representation

- Adjacency-list representation
  - generally good, especially for its optimal space complexity
  - bad for ***dense*** graphs and algorithms that require random access to edges
  - preferable for ***sparse*** graphs or graphs with ***low degree***

Choosing a Graph Representation

- Adjacency-list representation
  - generally good, especially for its optimal space complexity
  - bad for ***dense*** graphs and algorithms that require random access to edges
  - preferable for ***sparse*** graphs or graphs with ***low degree***
- Adjacency-matrix representation
  - suffers from a bad space complexity
  - good for algorithms that require random access to edges
  - preferable for ***dense*** graphs

Breadth-First Search

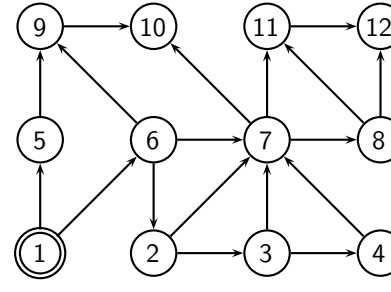
- One of the simplest but fundamental algorithms

Breadth-First Search

- One of the simplest but fundamental algorithms
- *Input:* $G = (V, E)$ and a vertex $s \in V$
  - explores the graph, touching all vertices that are reachable from s
  - iterates through the vertices at increasing distance (edge distance)
  - computes the distance of each vertex from s
  - produces a ***breadth-first tree*** rooted at s
  - works on both *directed* and *undirected* graphs

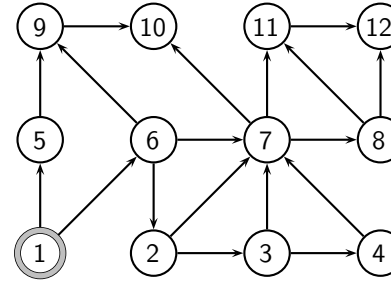
BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```



BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

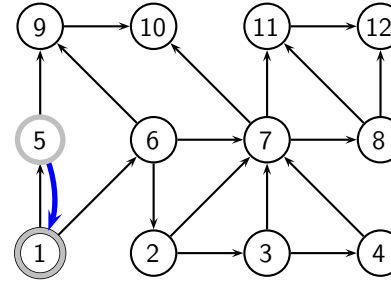


$u = 1$

$Q = \emptyset$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

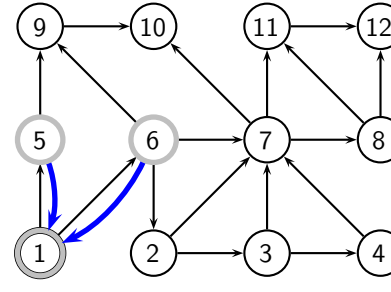


$u = 1$

$Q = \{5\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

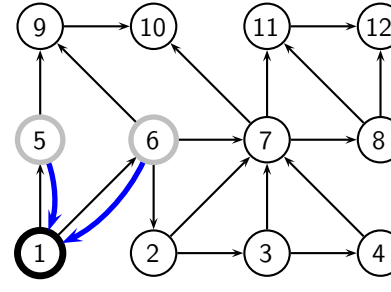


$u = 1$

$Q = \{5, 6\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

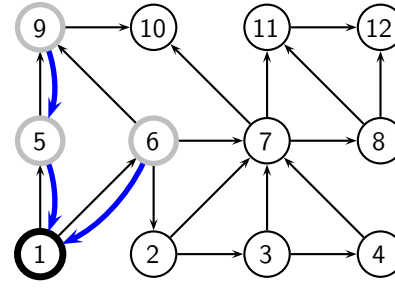


$u = 5$

$Q = \{6\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

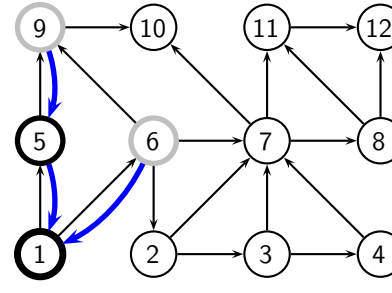


$u = 5$

$Q = \{6, 9\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

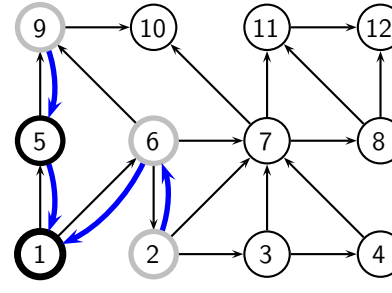


$u = 6$

$Q = \{9\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

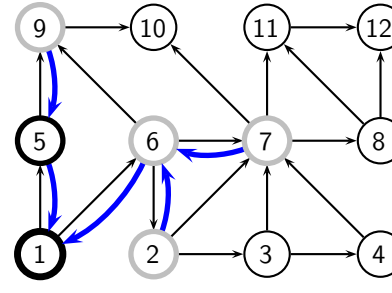


$u = 6$

$Q = \{9, 2, 7\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

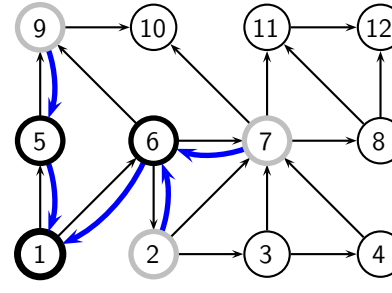


$u = 6$

$Q = \{9, 2, 7\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

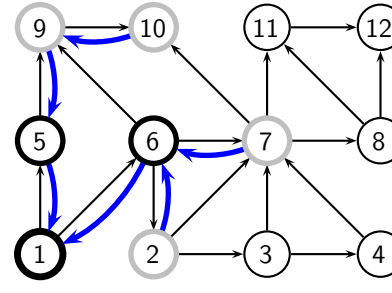


$u = 9$

$Q = \{2, 7\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

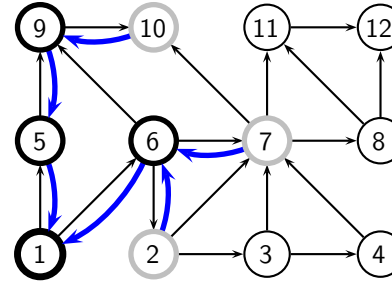


$u = 9$

$Q = \{2, 7, 10\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

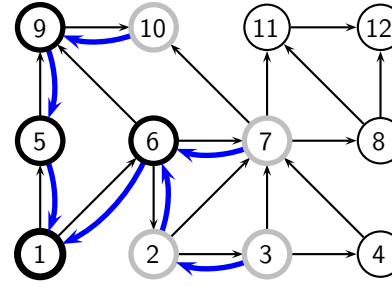


$u = 2$

$Q = \{7, 10\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

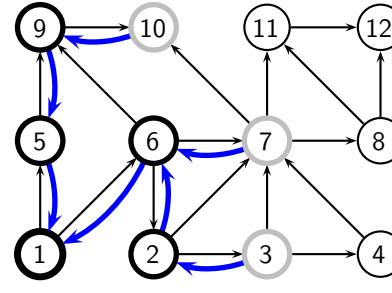


$u = 2$

$Q = \{7, 10, 3\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

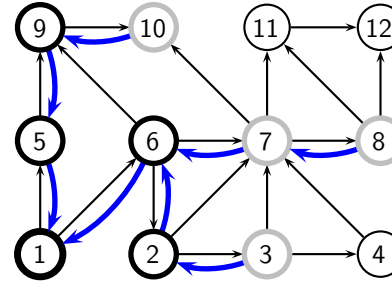


$u = 7$

$Q = \{10, 3\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2        color[u] = WHITE
           3        d[u] = ∞
           4        π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11        u = Dequeue(Q)
          12        for each v ∈ Adj[u]
          13             if color[v] == WHITE
          14                  color[v] = GRAY
          15                  d[v] = d[u] + 1
          16                  π[v] = u
          17                  Enqueue(Q, v)
          18        color[u] = BLACK
```

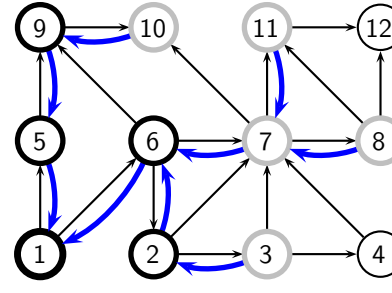


$u = 7$

$Q = \{10, 3, 8\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

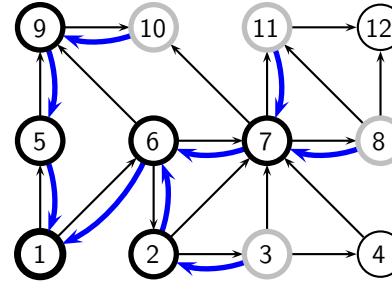


$u = 7$

$Q = \{10, 3, 8, 11\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

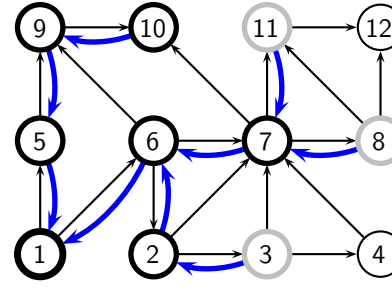


$u = 10$

$Q = \{3, 8, 11\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

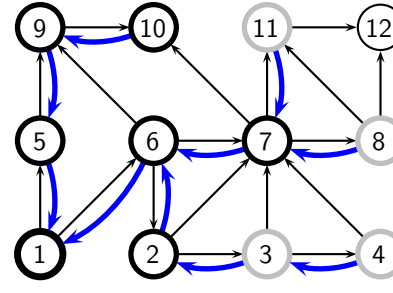


$u = 3$

$Q = \{8, 11\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

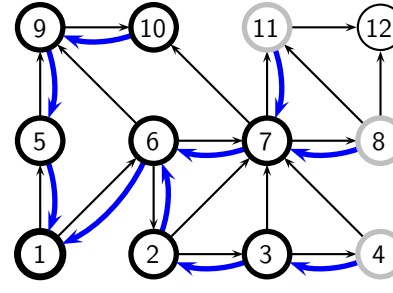


$u = 3$

$Q = \{8, 11, 4\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

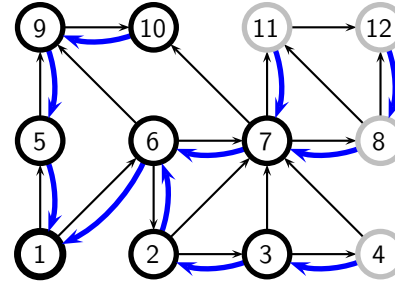


$u = 8$

$Q = \{11, 4\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

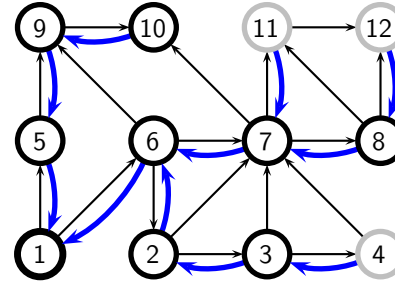


$u = 8$

$Q = \{11, 4, 12\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

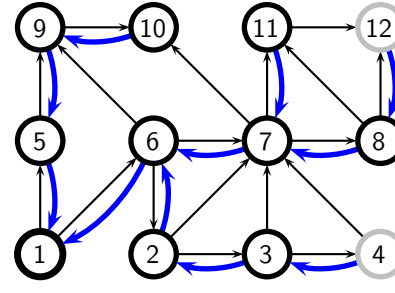


$u = 11$

$Q = \{4, 12\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

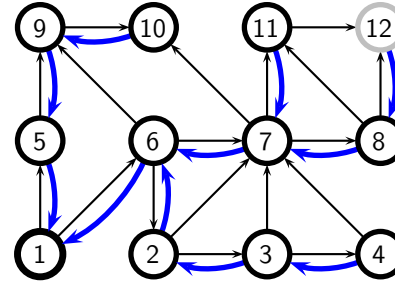


$u = 4$

$Q = \{12\}$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

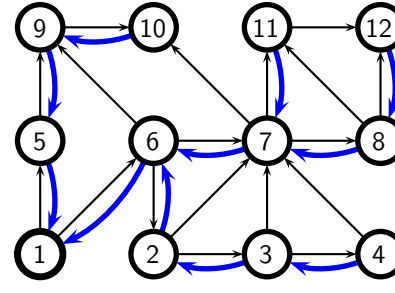


$u = 12$

$Q = \emptyset$

BFS Algorithm

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2        color[u] = WHITE
           3        d[u] = ∞
           4        π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11        u = Dequeue(Q)
          12        for each v ∈ Adj[u]
          13             if color[v] == WHITE
          14                  color[v] = GRAY
          15                  d[v] = d[u] + 1
          16                  π[v] = u
          17                  Enqueue(Q, v)
          18        color[u] = BLACK
```



Complexity of BFS

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

Complexity of BFS

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

- We enqueue a vertex only if it is white, and we immediately color it gray; thus, we enqueue every vertex *at most once*

Complexity of BFS

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

- We enqueue a vertex only if it is white, and we immediately color it gray; thus, we enqueue every vertex *at most once*
- So, the (dequeue) while loop executes $O(|V|)$ times

Complexity of BFS

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

- We enqueue a vertex only if it is white, and we immediately color it gray; thus, we enqueue every vertex *at most once*
- So, the (dequeue) while loop executes $O(|V|)$ times
- For each vertex u, the inner loop executes $\Theta(|E_u|)$, for a total of $O(|E|)$ steps

Complexity of BFS

```
BFS(G, s)  1  for each vertex u ∈ V(G) \ {s}
           2      color[u] = WHITE
           3      d[u] = ∞
           4      π[u] = NIL
           5  color[s] = GRAY
           6  d[s] = 0
           7  π[s] = NIL
           8  Q = ∅
           9  Enqueue(Q, s)
          10  while Q ≠ ∅
          11      u = Dequeue(Q)
          12      for each v ∈ Adj[u]
          13          if color[v] == WHITE
          14              color[v] = GRAY
          15              d[v] = d[u] + 1
          16              π[v] = u
          17              Enqueue(Q, v)
          18      color[u] = BLACK
```

- We enqueue a vertex only if it is white, and we immediately color it gray; thus, we enqueue every vertex *at most once*
- So, the (dequeue) while loop executes $O(|V|)$ times
- For each vertex u, the inner loop executes $\Theta(|E_u|)$, for a total of $O(|E|)$ steps
- So, $O(|V| + |E|)$

Depth-First Search

Depth-First Search

- Immediately follow the links of the most recently-visited vertex, then backtrack when you reach a dead-end
  - i.e., backtrack when the current vertex has no more adjacent vertices that have not yet been visited

Depth-First Search

- Immediately follow the links of the most recently-visited vertex, then backtrack when you reach a dead-end
  - i.e., backtrack when the current vertex has no more adjacent vertices that have not yet been visited

- *Input:* $G = (V, E)$
  - explores the graph, touching *all vertices*

Depth-First Search

- Immediately follow the links of the most recently-visited vertex, then backtrack when you reach a dead-end
  - i.e., backtrack when the current vertex has no more adjacent vertices that have not yet been visited

- *Input:* $G = (V, E)$
  - explores the graph, touching *all vertices*
  - produces a ***depth-first forest***, consisting of all the ***depth-first trees*** defined by the DFS exploration

Depth-First Search

- Immediately follow the links of the most recently-visited vertex, then backtrack when you reach a dead-end
  - i.e., backtrack when the current vertex has no more adjacent vertices that have not yet been visited

- *Input:* $G = (V, E)$
  - explores the graph, touching *all vertices*
  - produces a ***depth-first forest***, consisting of all the ***depth-first trees*** defined by the DFS exploration
  - associates ***two time-stamps*** to each vertex
    - $d[u]$ records when u is first discovered
    - $f[u]$ records when DFS finishes examining u's edges, and therefore backtracks from u

DFS Algorithm

```
DFS(G)1  for each vertex u ∈ V(G)
2            color[u] = WHITE
3            π[u] = NIL
4        time = 0 // "global" variable
5        for each vertex u ∈ V(G)
6            if color[u] == WHITE
7                DFS-Visit(u)
```

```
DFS-Visit(u)  1   color[u] = GREY
              2   time = time + 1
              3   d[u] = time
              4   for each v ∈ Adj[u]
              5       if color[v] == WHITE
              6           π[v] = u
              7           DFS-Visit(v)
              8   color[u] = BLACK
              9   time = time + 1
             10   f[u] = time
```

Complexity of DFS

Complexity of DFS

- The loop in **DFS-Visit**(u) (lines 4–7) accounts for $\Theta(|E_u|)$

Complexity of DFS

- The loop in **DFS-Visit**(u) (lines 4–7) accounts for $\Theta(|E_u|)$
- We call **DFS-Visit**(u) *once* for each vertex u
  - either in **DFS**, or recursively in **DFS-Visit**
  - because we call it only if $color[u] = \text{WHITE}$, but then we immediately set $color[u] = \text{GREY}$

Complexity of DFS

- The loop in **DFS-Visit**(u) (lines 4–7) accounts for $\Theta(|E_u|)$
- We call **DFS-Visit**(u) *once* for each vertex u
  - either in **DFS**, or recursively in **DFS-Visit**
  - because we call it only if $color[u] = \text{WHITE}$, but then we immediately set $color[u] = \text{GREY}$
- So, the overall complexity is $\Theta(|V| + |E|)$

Applications of DFS: Topological Sort

Applications of DFS: Topological Sort

- **Problem:** (topological sort)

  Given a *directed acyclic graph* (DAG)

  - find an ordering of vertices such that you only end up with *forward links*

Applications of DFS: Topological Sort

- **Problem:** (topological sort)

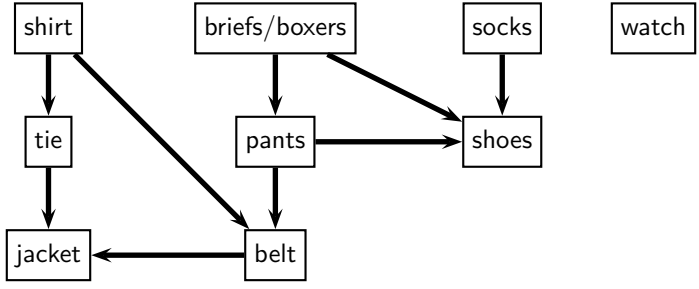
  Given a *directed acyclic graph* (DAG)

  - find an ordering of vertices such that you only end up with *forward links*

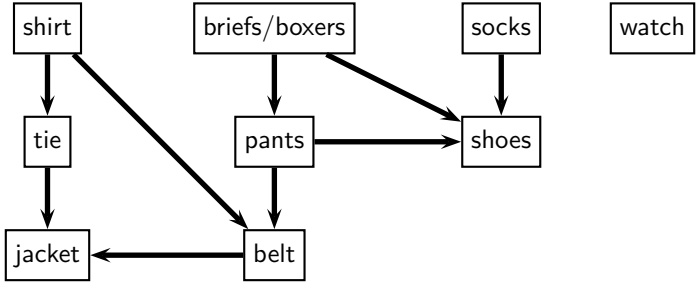
- **Example:** dependencies in software packages
  - find an installation order for a set of software packages
  - such that every package is installed only after all the packages it depends on

Topological Sort Algorithm

Topological Sort Algorithm



Topological Sort Algorithm



Topological-Sort(G) 1 **DFS**(G)
2 output V sorted in reverse order of $f[\cdot]$