Grafer, traversering

Koffman & Wolfgang

- kapitel 10, avsnitt 4

Traversering av grafer

De flesta grafalgoritmer innebär att besöka varje nod i någon systematisk ordning

- precis som med träd så finns det olika sätt att göra detta på

De två vanligaste metoderna är:

- bredden-först-sökning
- djupet-först-sökning

BFS: Bredden-först-sökning

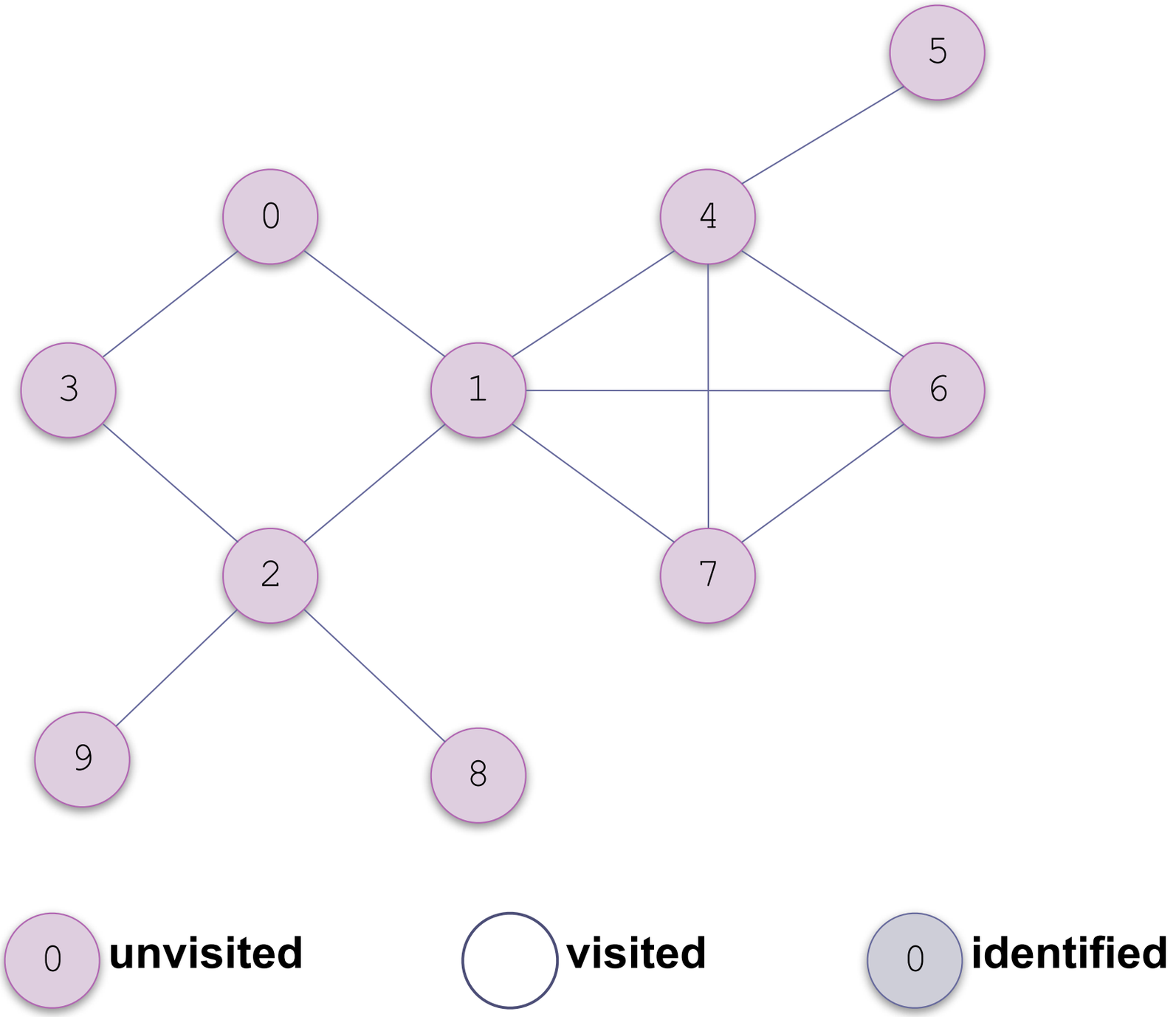
Vid bredden-först så besöker vi noderna i följande ordning:

- besök startnoden först
- sedan alla angränsande noder
- sedan alla noder som kan nås via två bågar
- sedan alla noder som kan nås via tre bågar
- och så vidare

Vi besöker alltså alla noder som kan nås i k steg, innan vi besöker de noder som kan nås i k+1 steg.

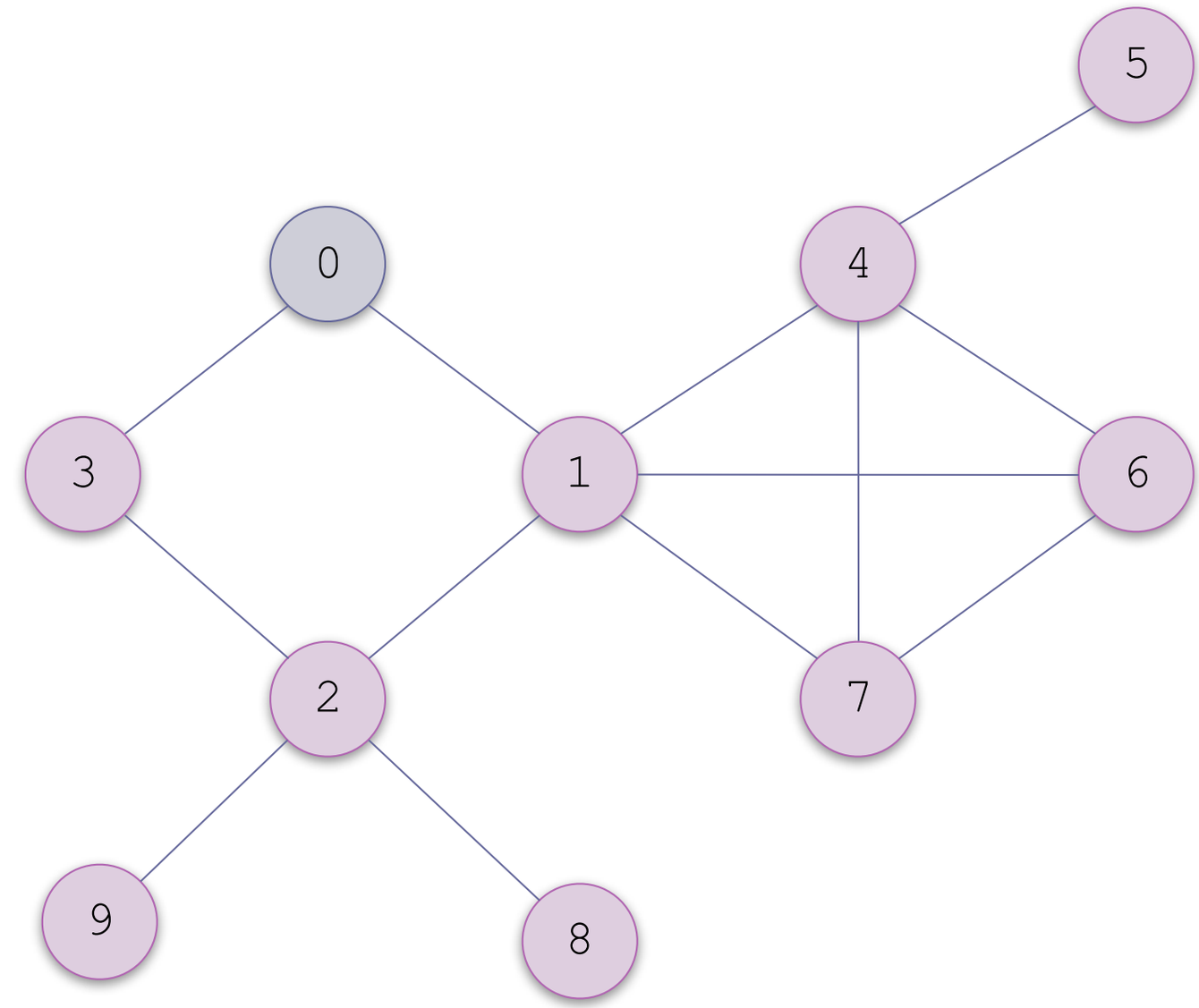
Eftersom ingen nod är speciell så antar vi för enkelhets skull att nod nr 0 är startnoden.

Example of a Breadth-First Search



Example of a Breadth-First Search (cont.)

Identify the start node



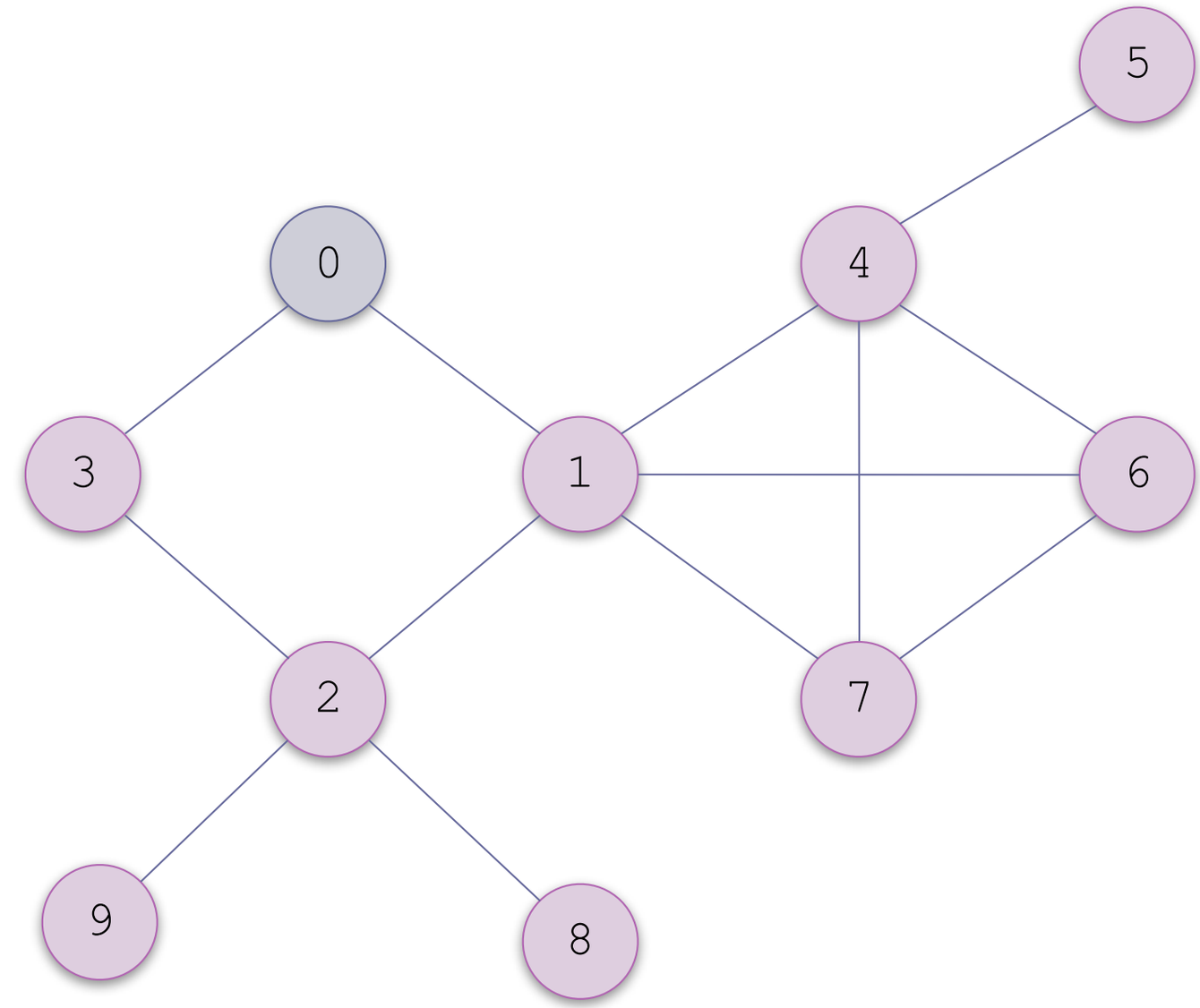
0 **unvisited**

visited

0 **identified**

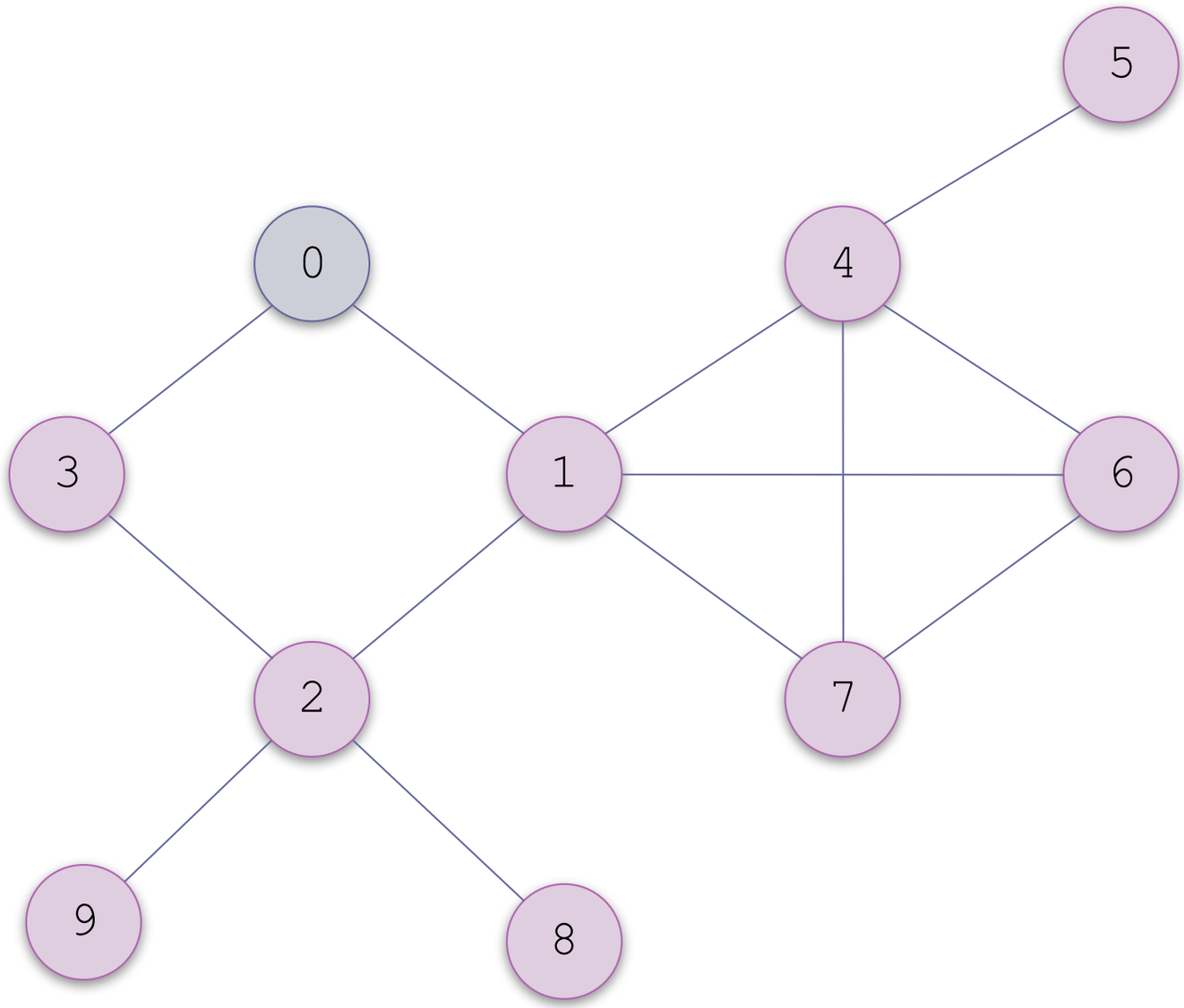
Example of a Breadth-First Search (cont.)

While visiting it, we can identify its adjacent nodes



Example of a Breadth-First Search (cont.)

We identify its adjacent nodes and add them to a queue of identified nodes



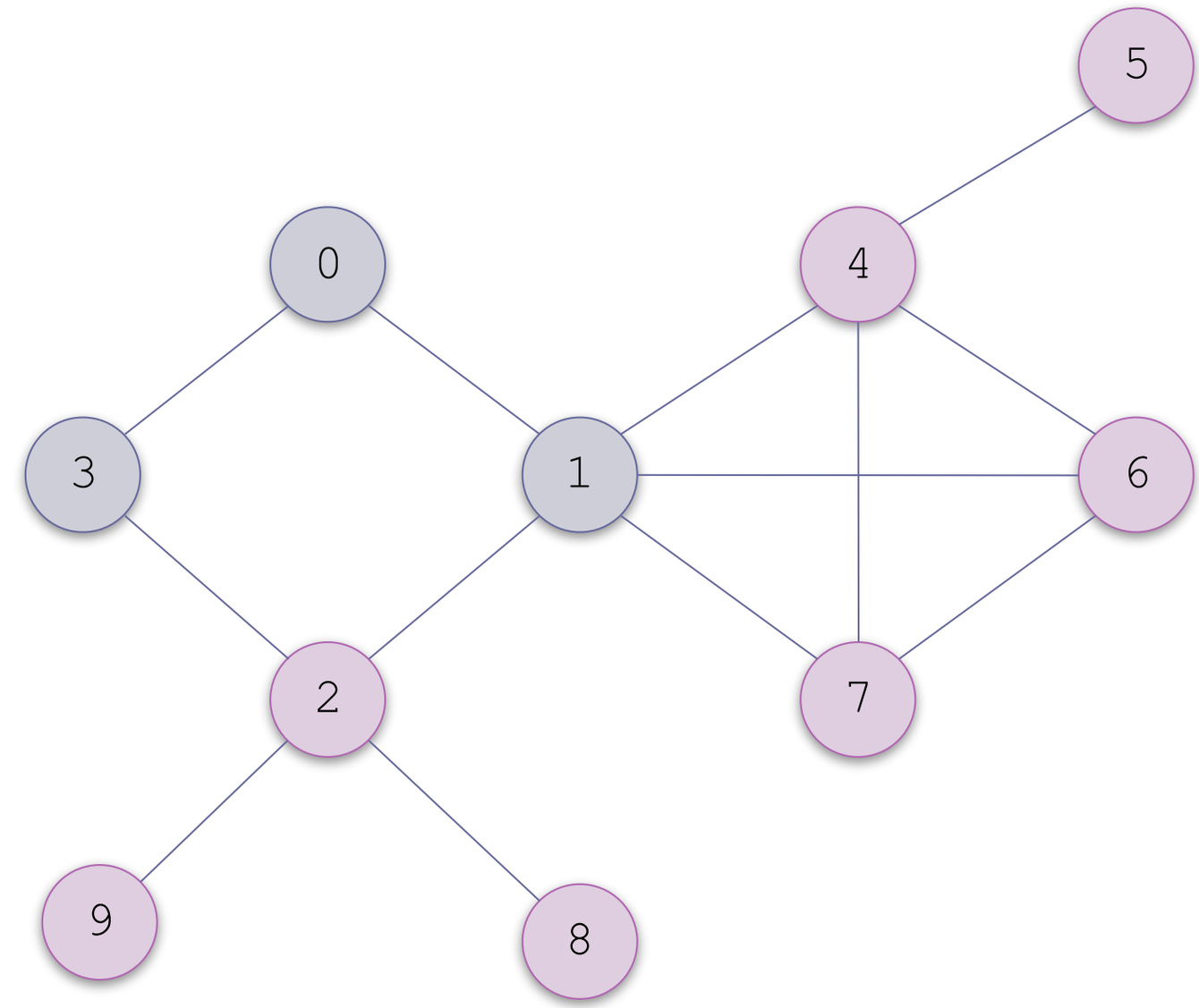
Visit sequence:
0

Example of a Breadth-First Search (cont.)

We identify its adjacent nodes and add them to a queue of identified nodes

Queue:
1, 3

Visit sequence:
0

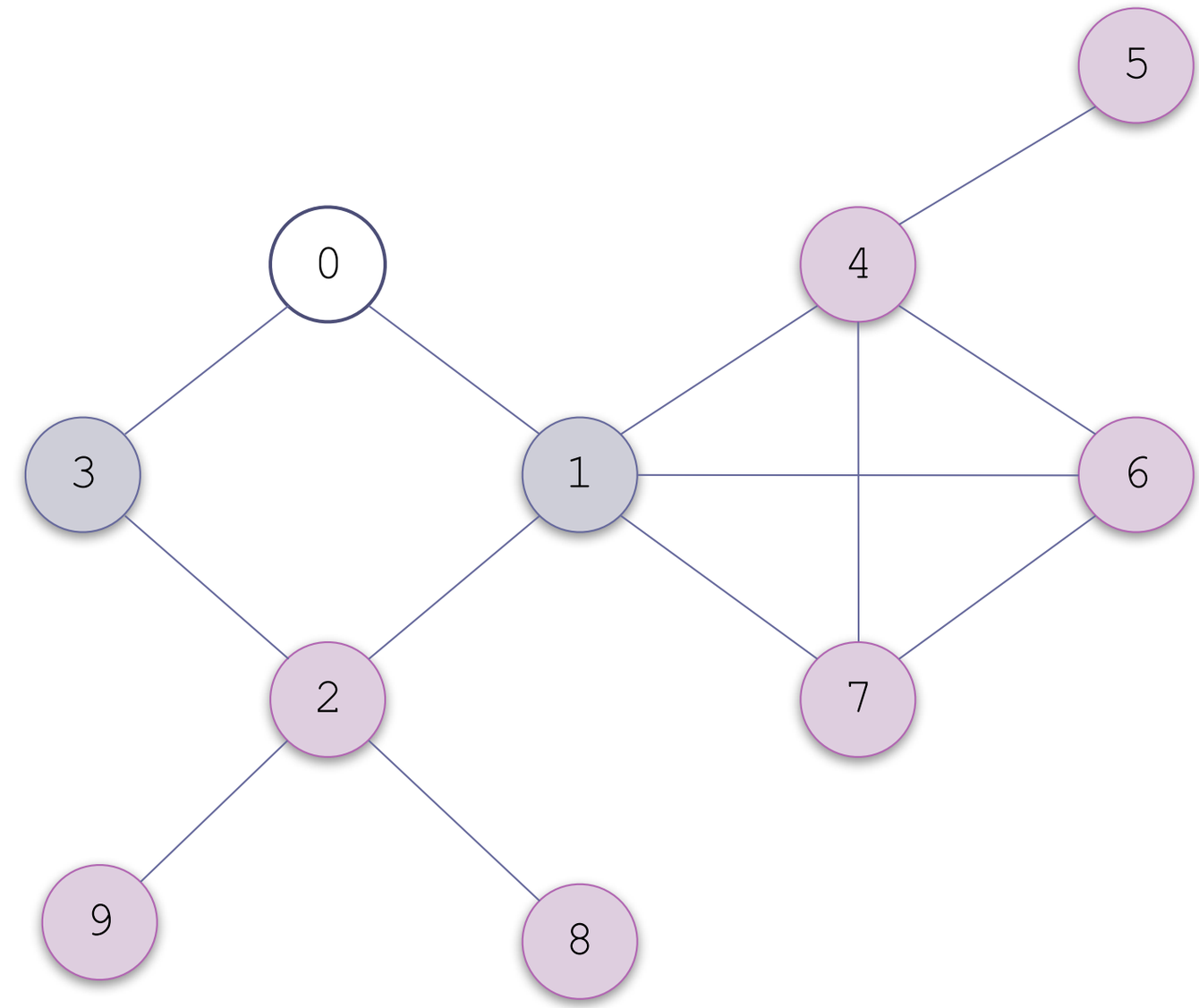


Example of a Breadth-First Search (cont.)

We color the node as visited

Queue:
1, 3

Visit sequence:
0

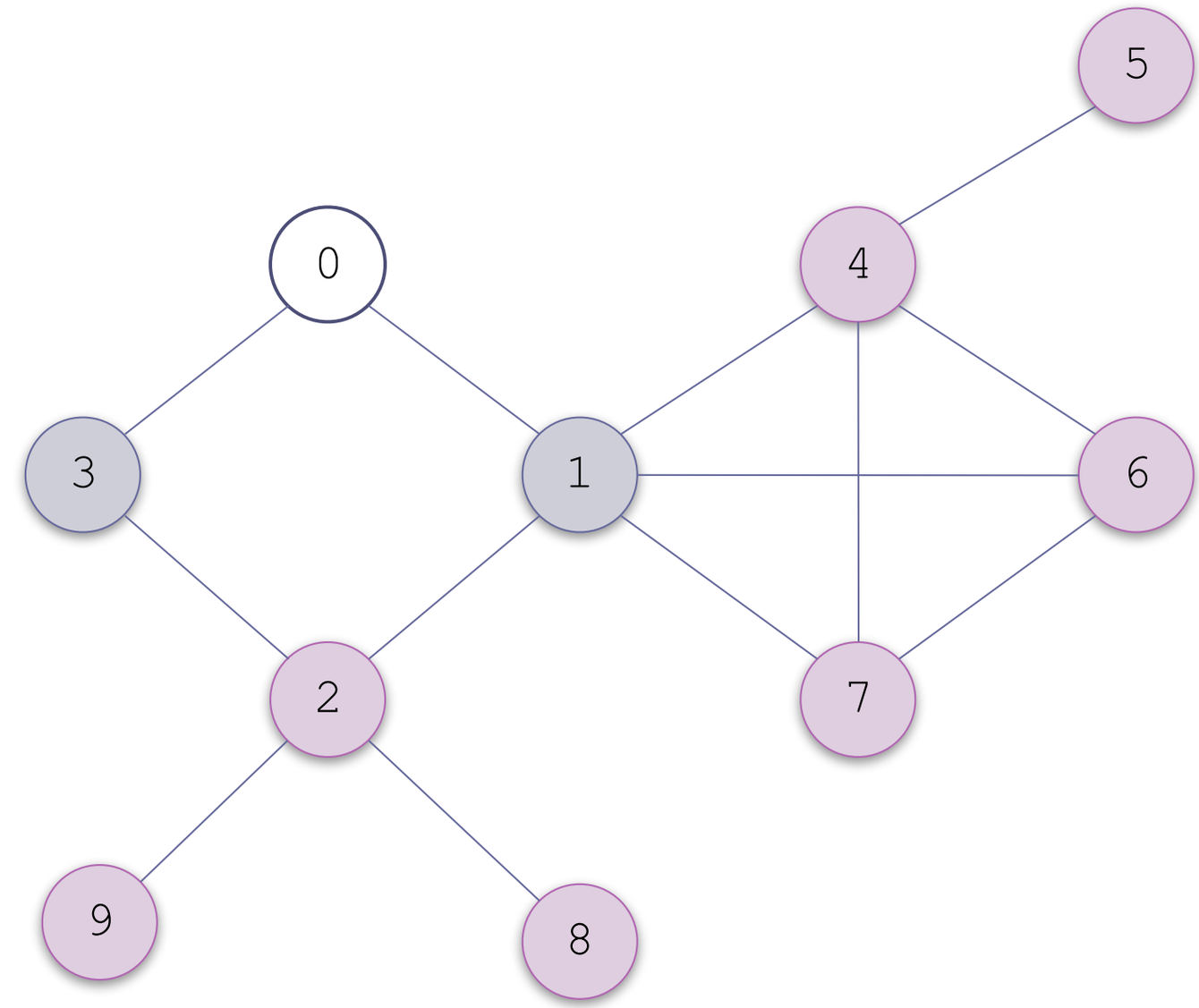


Example of a Breadth-First Search (cont.)

The queue determines which nodes to visit next

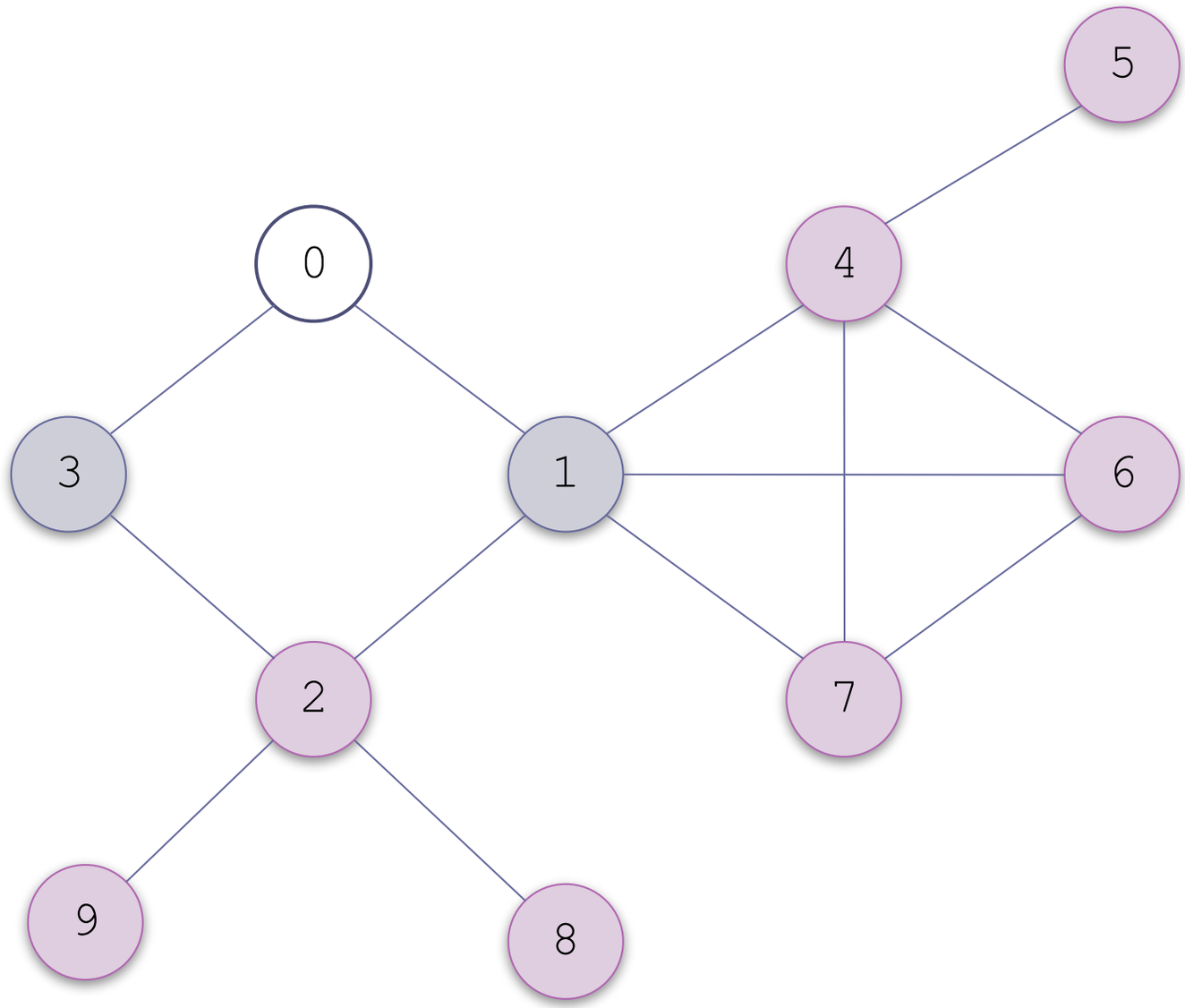
Queue:
1, 3

Visit sequence:
0



Example of a Breadth-First Search (cont.)

Visit the first node in the queue, 1

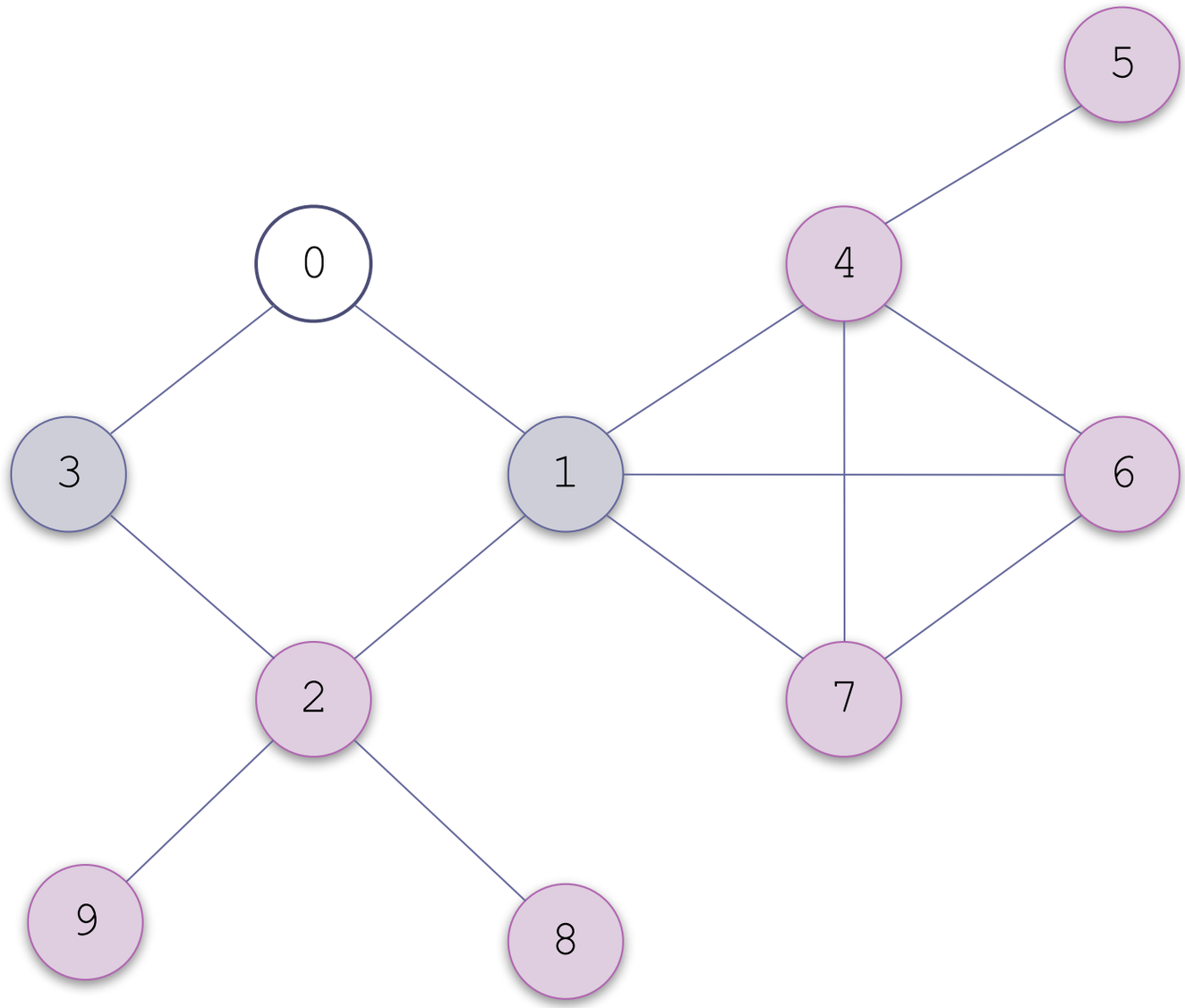


Queue:
1, 3

Visit sequence:
0

Example of a Breadth-First Search (cont.)

Visit the first node in the queue, 1



Queue:
3

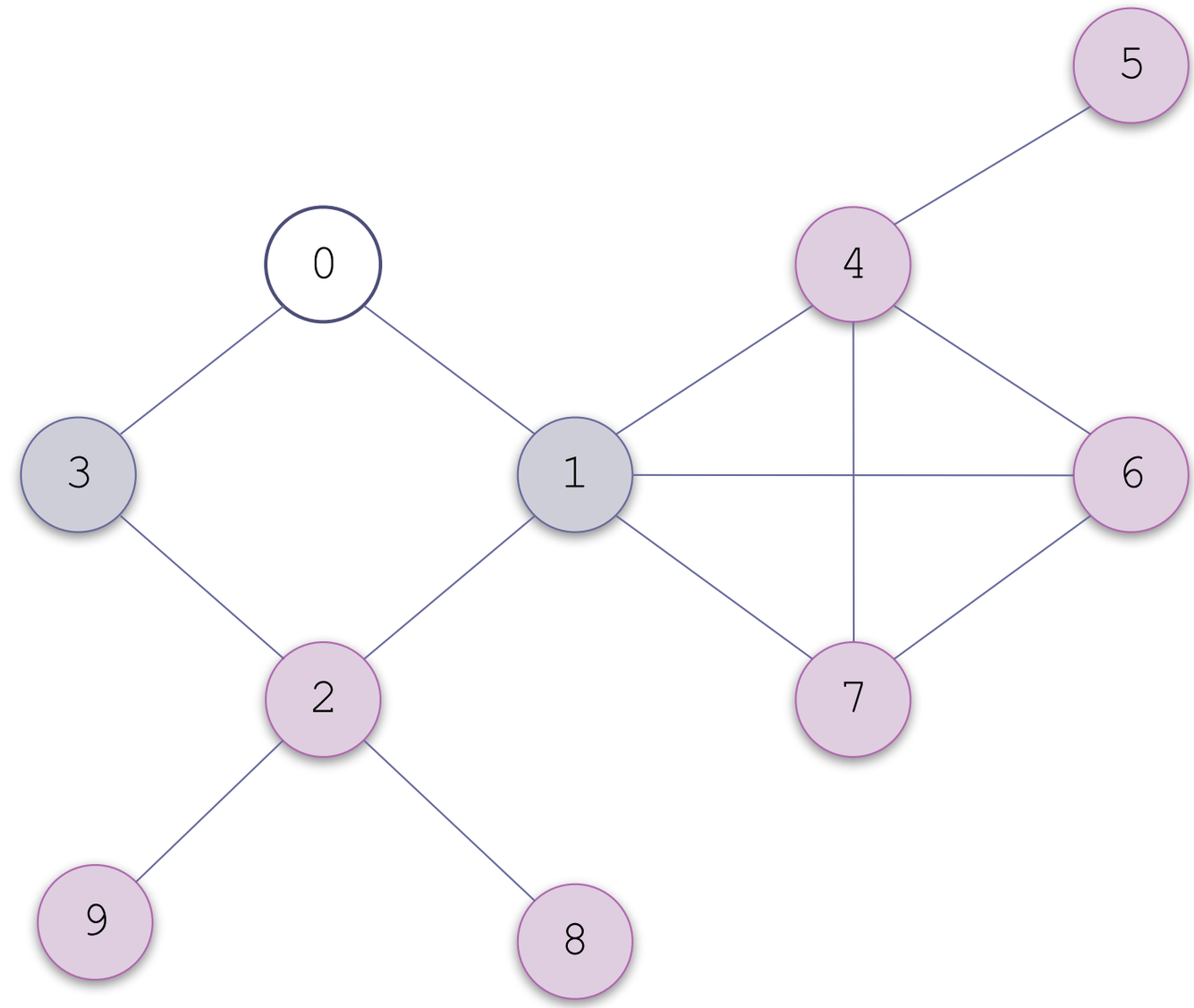
Visit sequence:
0, 1

Example of a Breadth-First Search (cont.)

Select all its adjacent nodes that have not been visited or identified

Queue:
3

Visit sequence:
0, 1

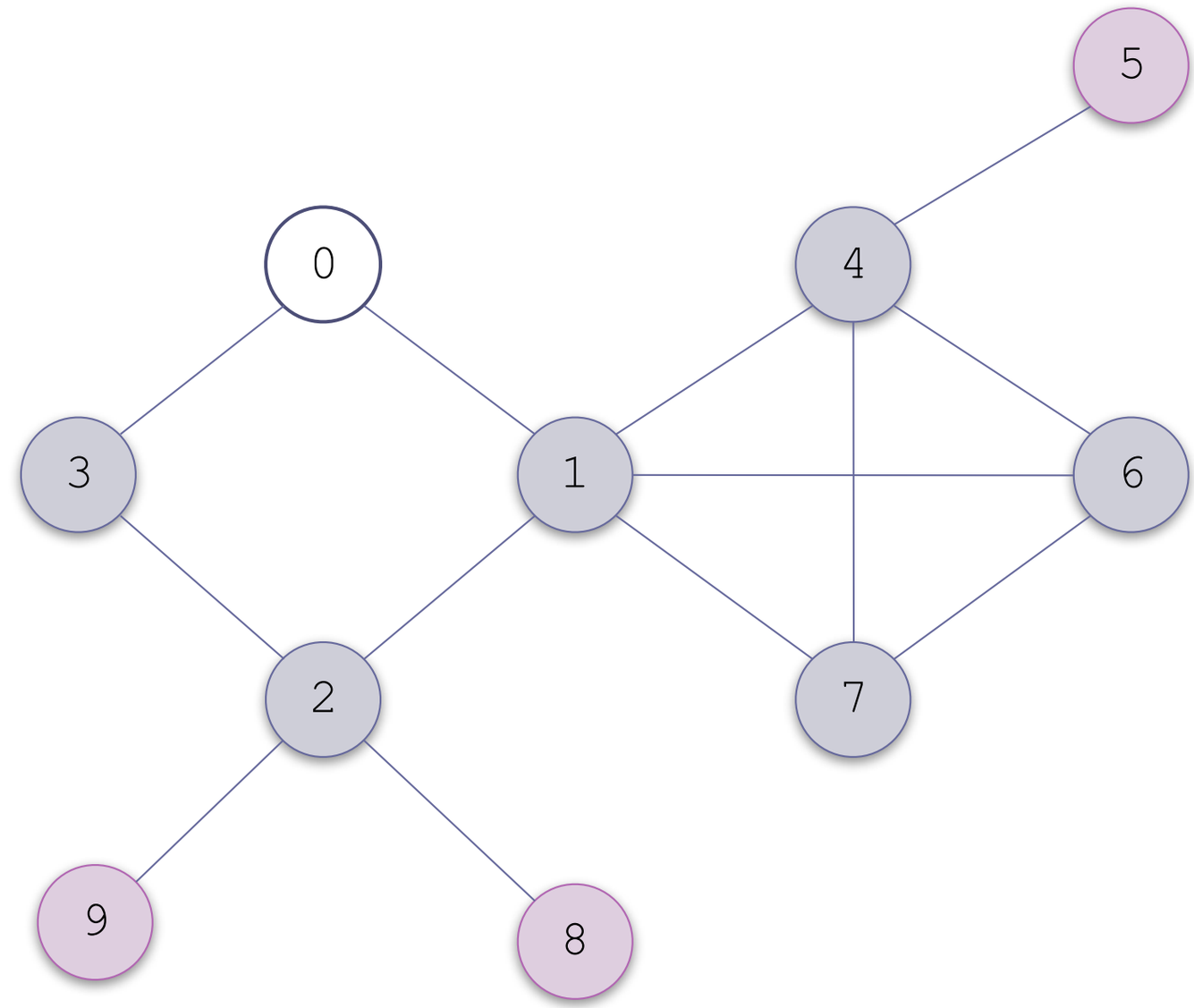


Example of a Breadth-First Search (cont.)

Select all its adjacent nodes that have not been visited or identified

Queue:
3, 2, 4, 6, 7

Visit sequence:
0, 1

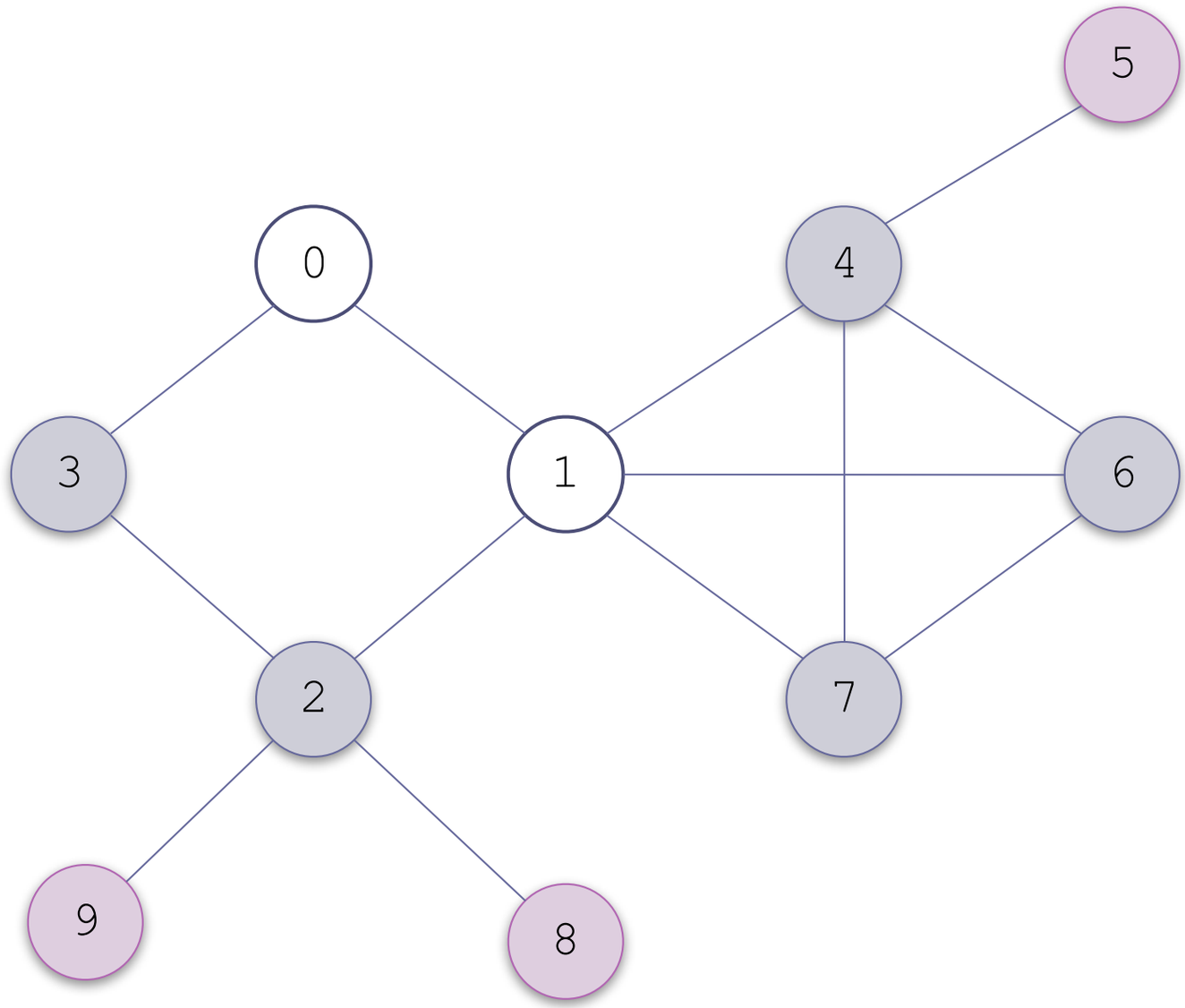


Example of a Breadth-First Search (cont.)

Now that we are done with 1, we color it as visited

Queue:
3, 2, 4, 6, 7

Visit sequence:
0, 1



unvisited

visited

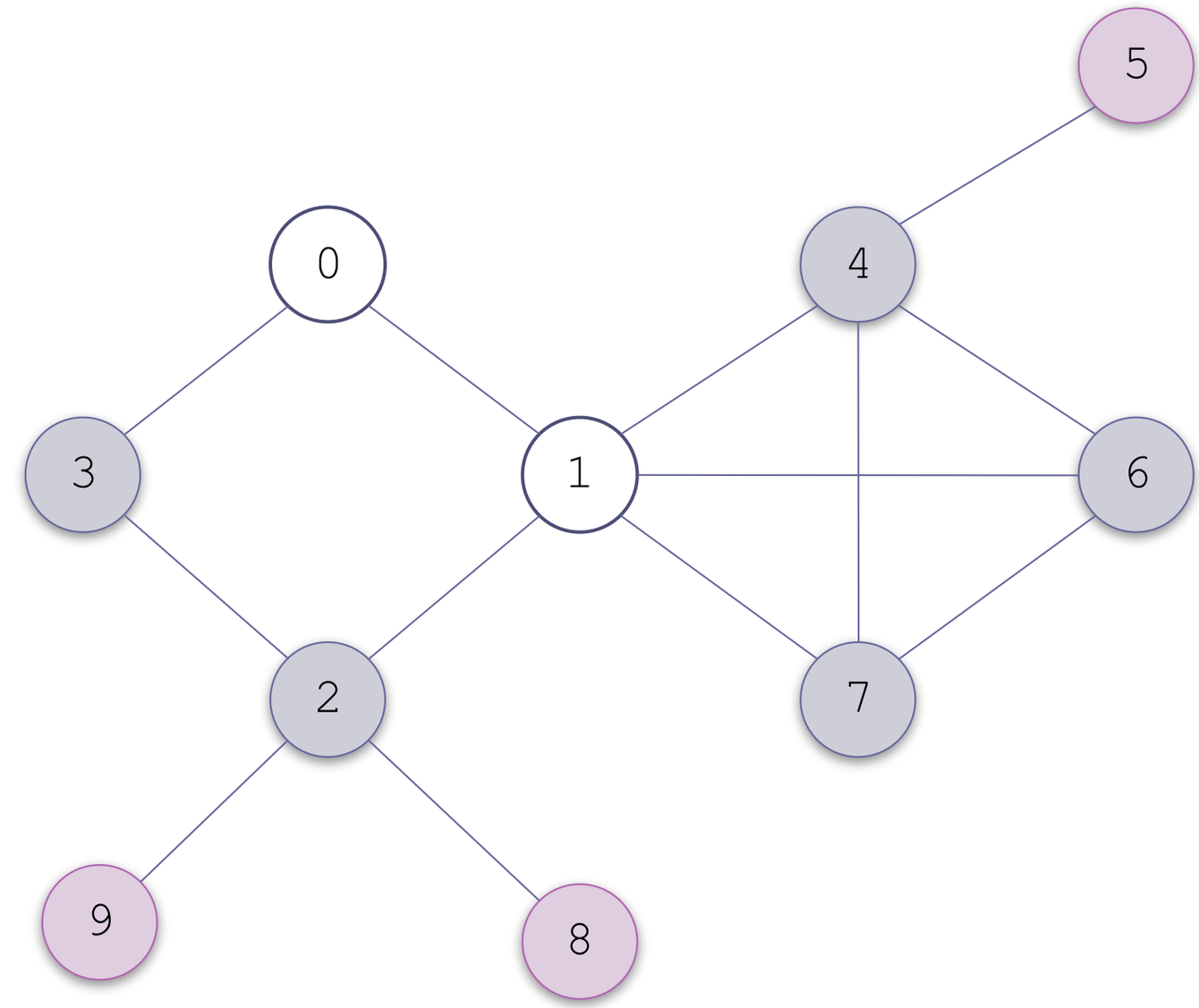
identified

Example of a Breadth-First Search (cont.)

and then visit the next node in the queue, 3 (which was identified in the first selection)

Queue:
3, 2, 4, 6, 7

Visit sequence:
0, 1

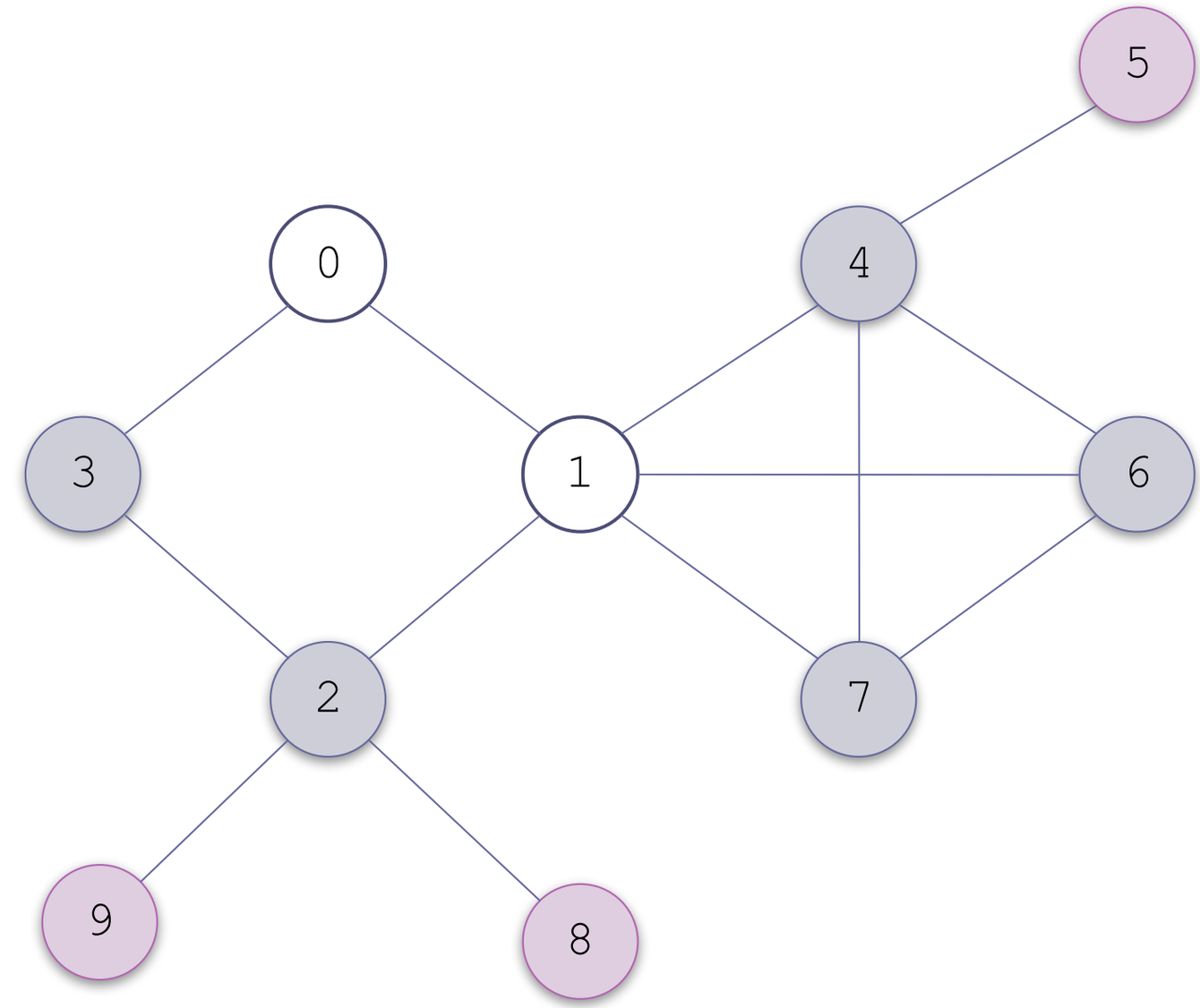


Example of a Breadth-First Search (cont.)

and then visit the next node in the queue, 3 (which was identified in the first selection)

Queue:
2, 4, 6, 7

Visit sequence:
0, 1, 3

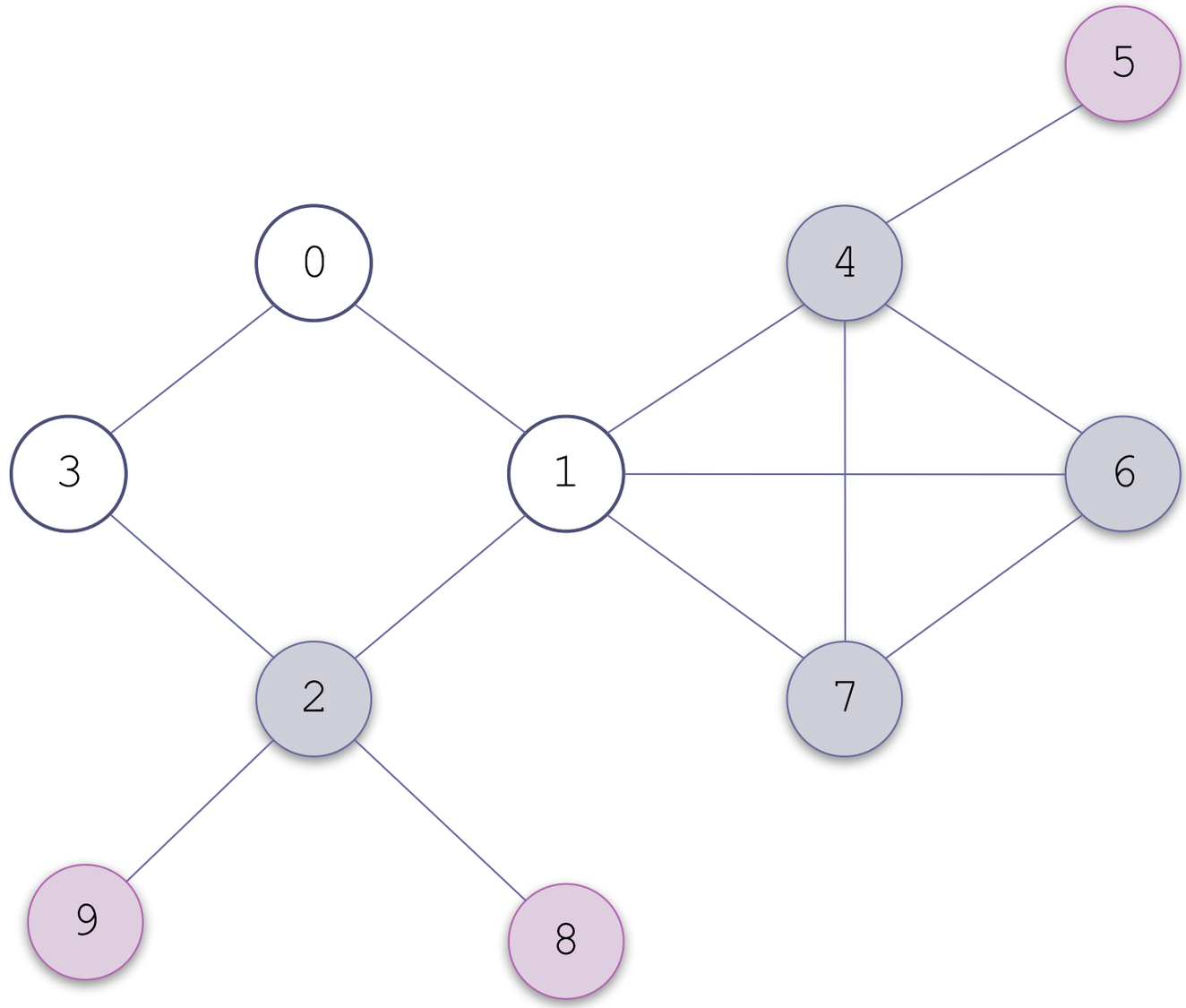


Example of a Breadth-First Search (cont.)

3 has two adjacent vertices. 0 has already been visited and 2 has already been identified. We are done with 3

Queue:
2, 4, 6, 7

Visit sequence:
0, 1, 3



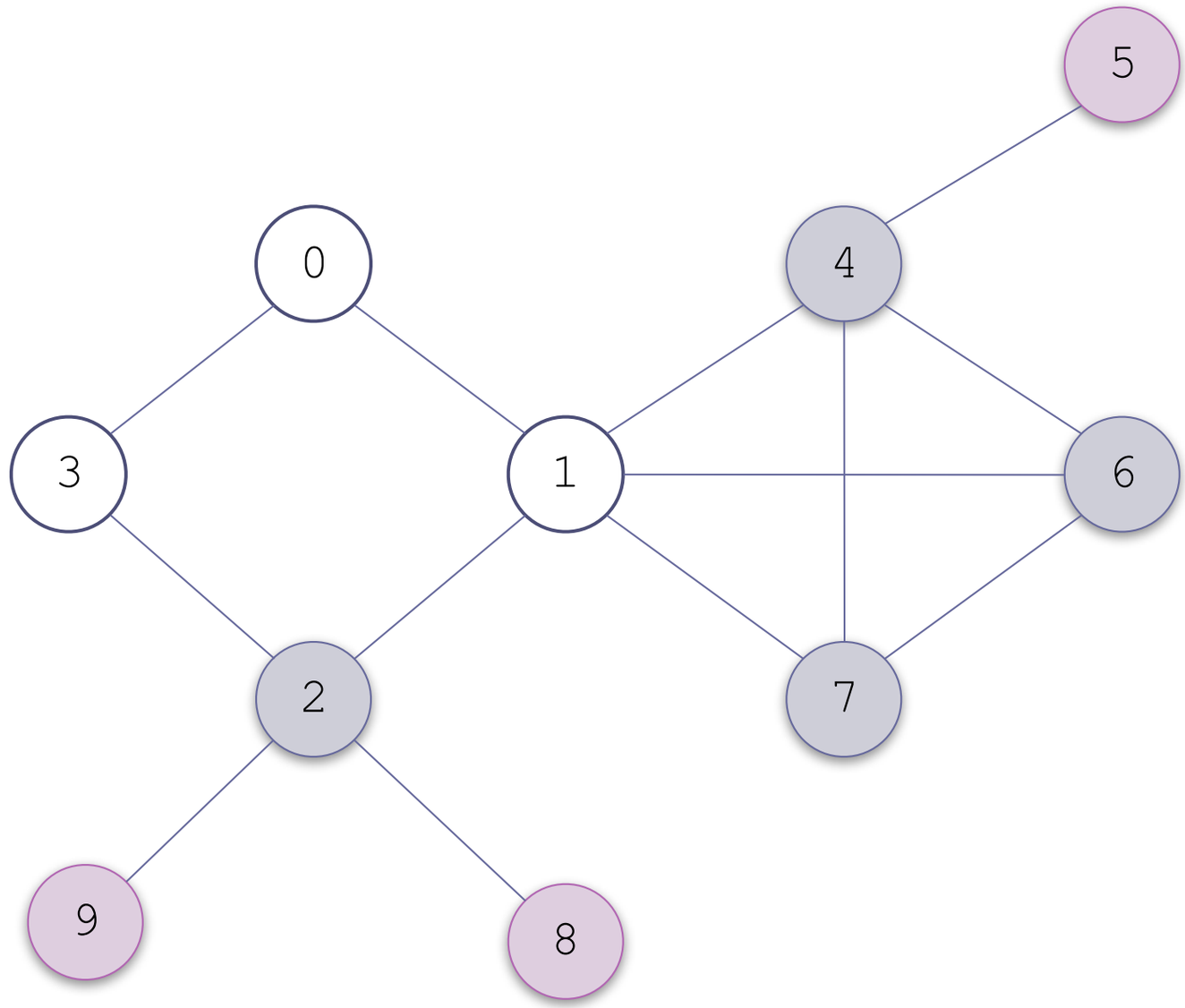
unvisited

visited

identified

Example of a Breadth-First Search (cont.)

The next node in the queue is 2



Queue:
2, 4, 6, 7

Visit sequence:
0, 1, 3

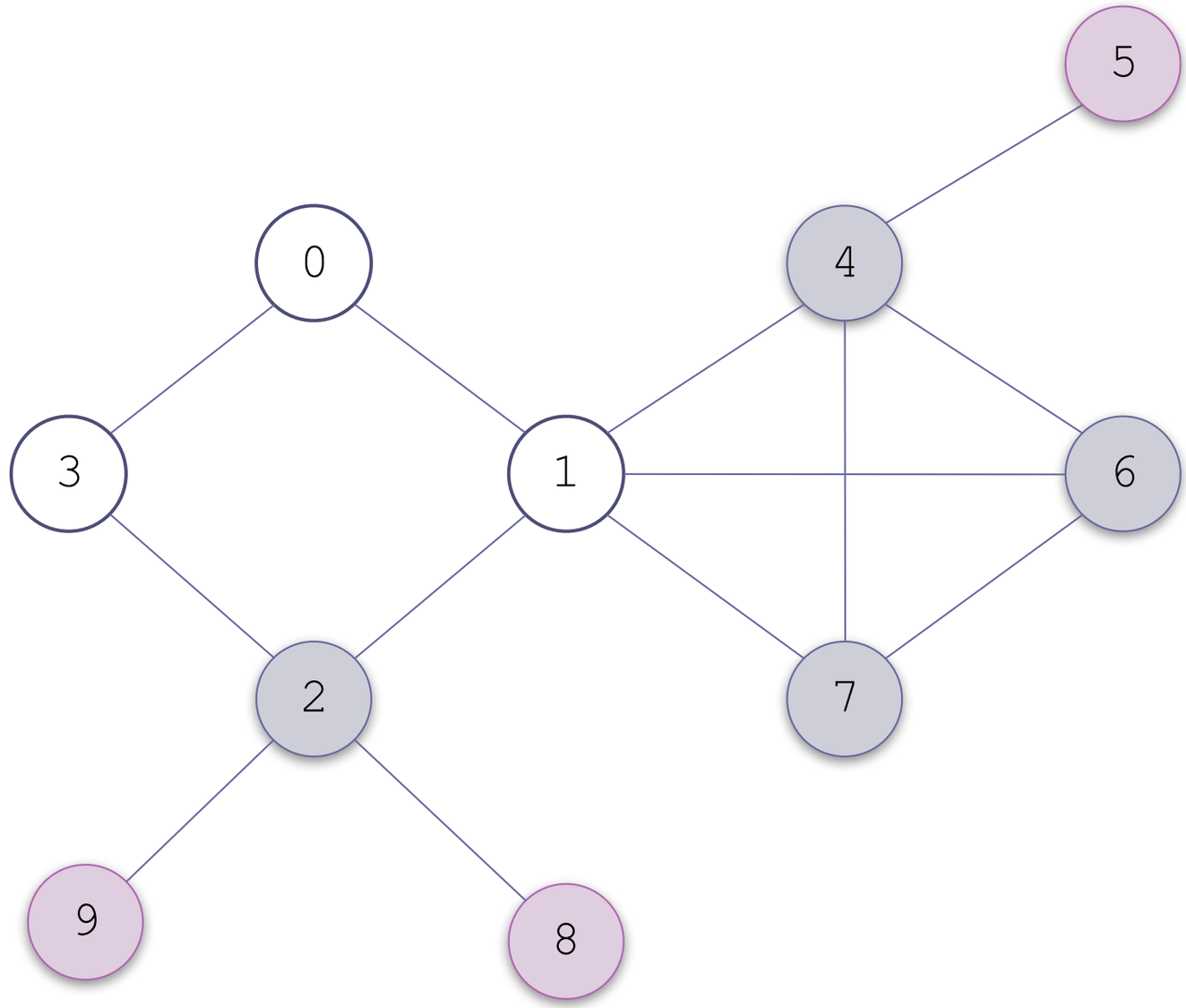
0 **unvisited**

visited

0 **identified**

Example of a Breadth-First Search (cont.)

The next node in the queue is 2



Queue:
4, 6, 7

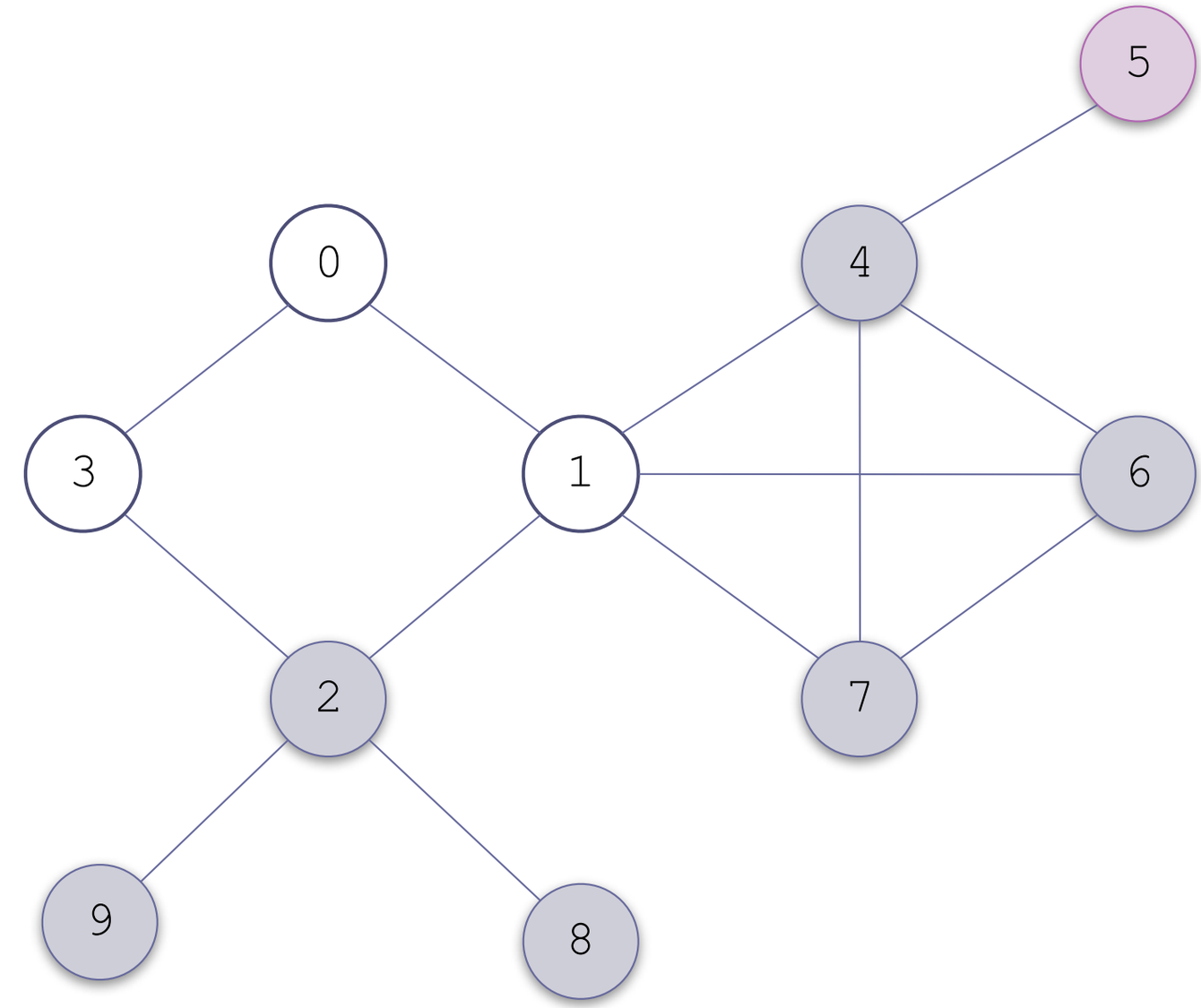
Visit sequence:
0, 1, 3, 2

Example of a Breadth-First Search (cont.)

8 and 9 are the only adjacent vertices not already visited or identified

Queue:
4, 6, 7, 8, 9

Visit sequence:
0, 1, 3, 2

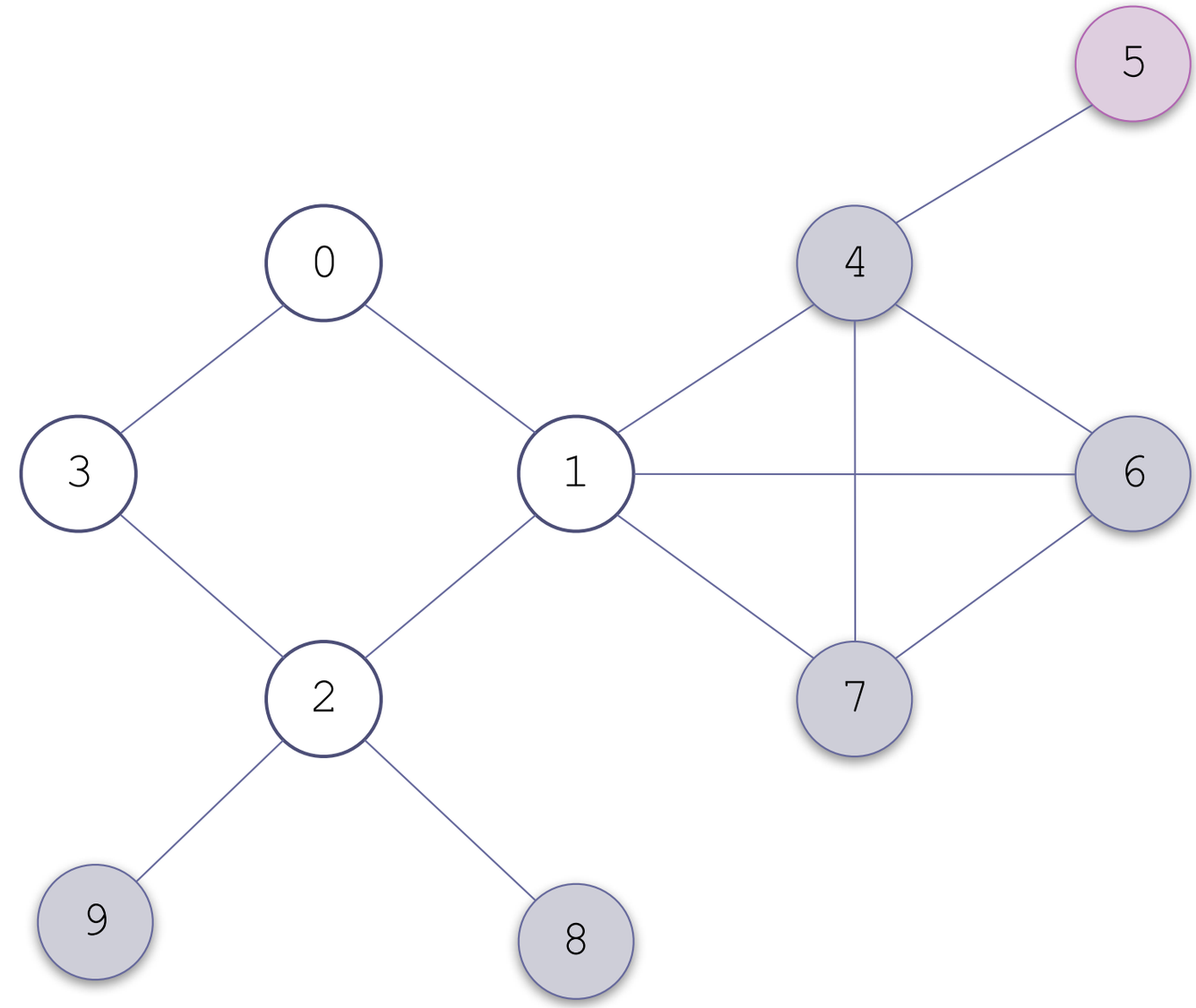


Example of a Breadth-First Search (cont.)

4 is next

Queue:
6, 7, 8, 9

Visit sequence:
0, 1, 3, 2, 4

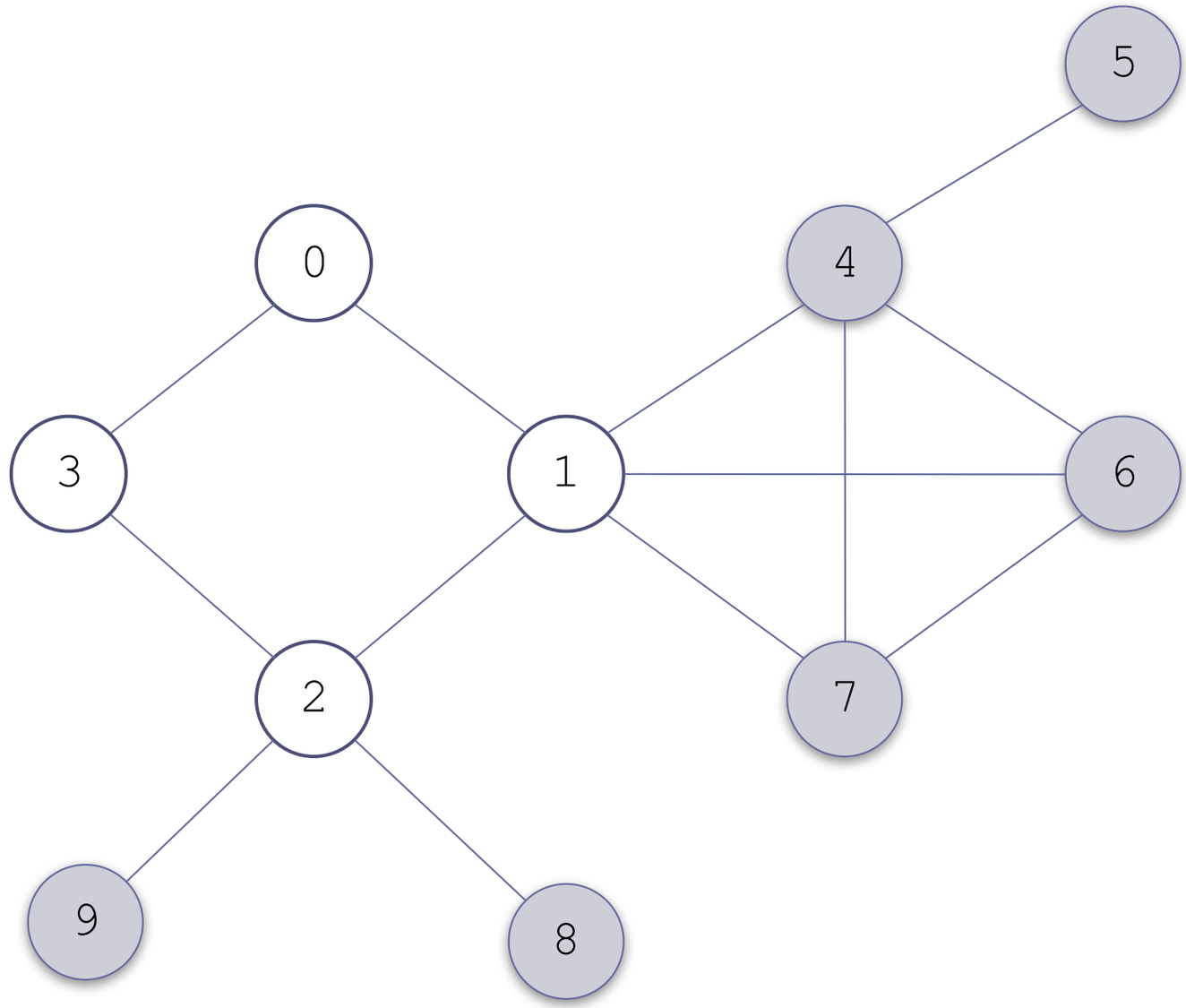


Example of a Breadth-First Search (cont.)

5 is the only vertex not already visited or identified

Queue:
6, 7, 8, 9, 5

Visit sequence:
0, 1, 3, 2, 4

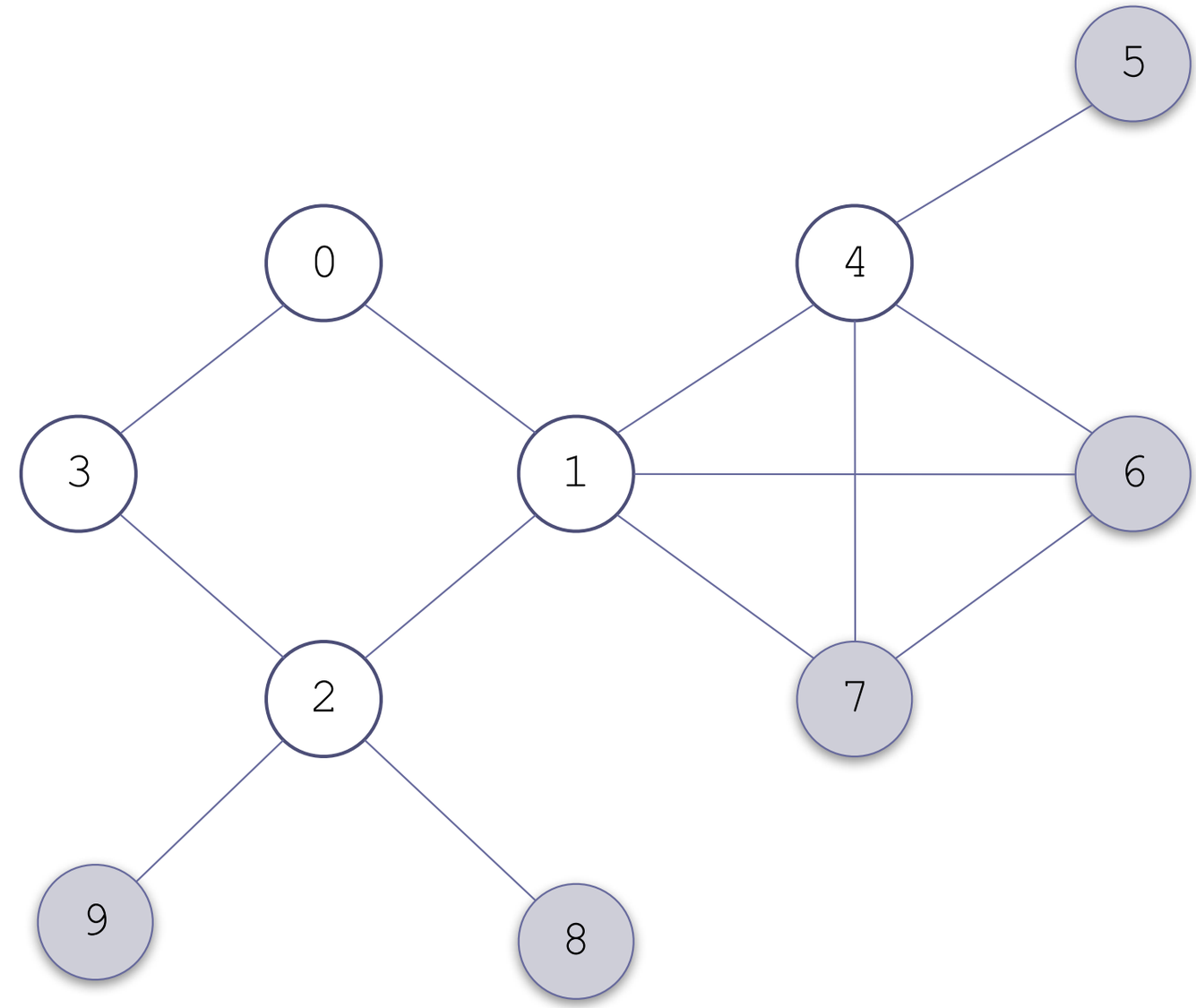


Example of a Breadth-First Search (cont.)

6 has no vertices not already visited or identified

Queue:
7, 8, 9, 5

Visit sequence:
0, 1, 3, 2, 4, 6

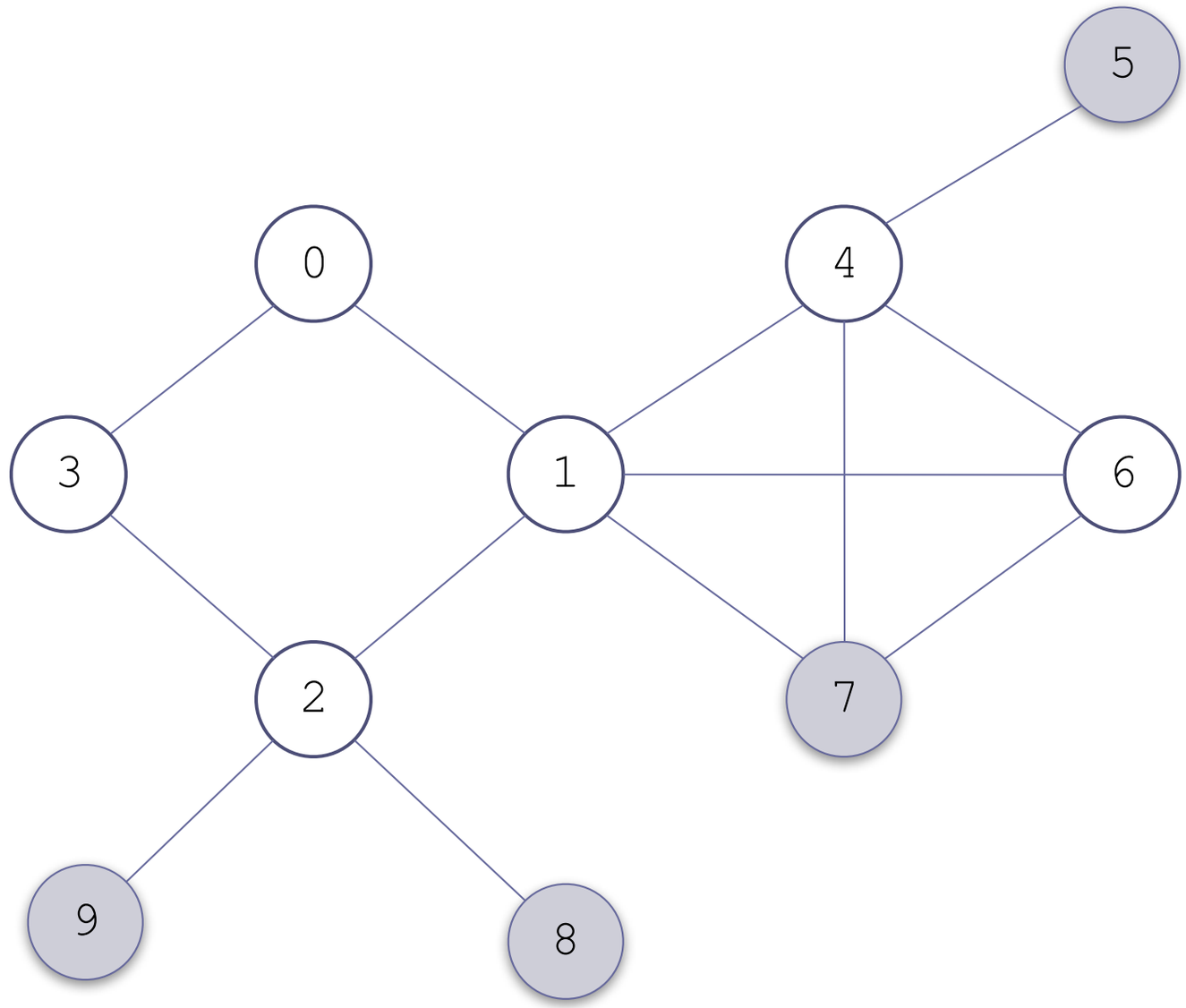


Example of a Breadth-First Search (cont.)

6 has no vertices not already visited or identified

Queue:
7, 8, 9, 5

Visit sequence:
0, 1, 3, 2, 4, 6

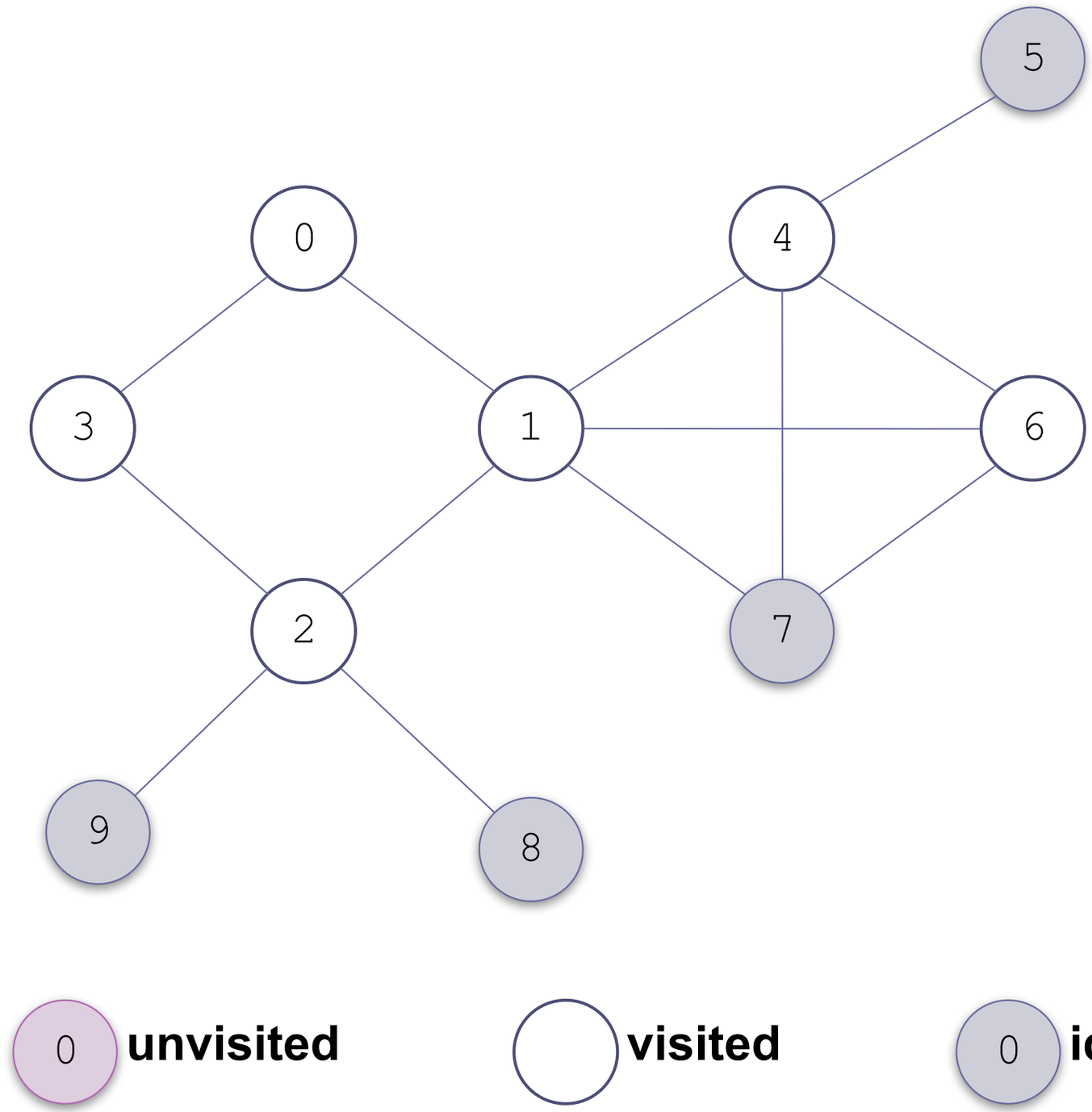


Example of a Breadth-First Search (cont.)

7 has no vertices not already visited or identified

Queue:
8, 9, 5

Visit sequence:
0, 1, 3, 2, 4, 6, 7

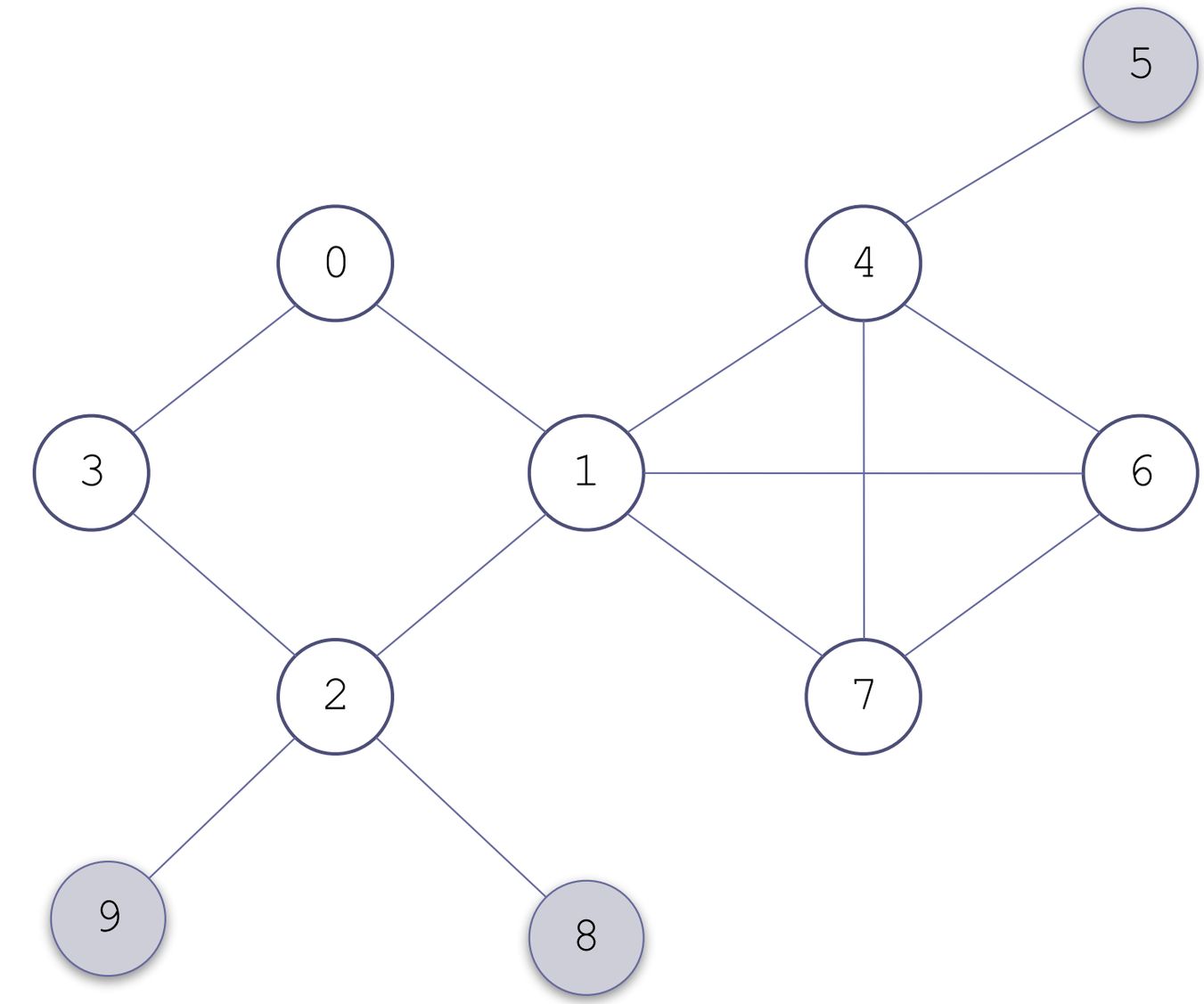


Example of a Breadth-First Search (cont.)

7 has no vertices not already visited or identified

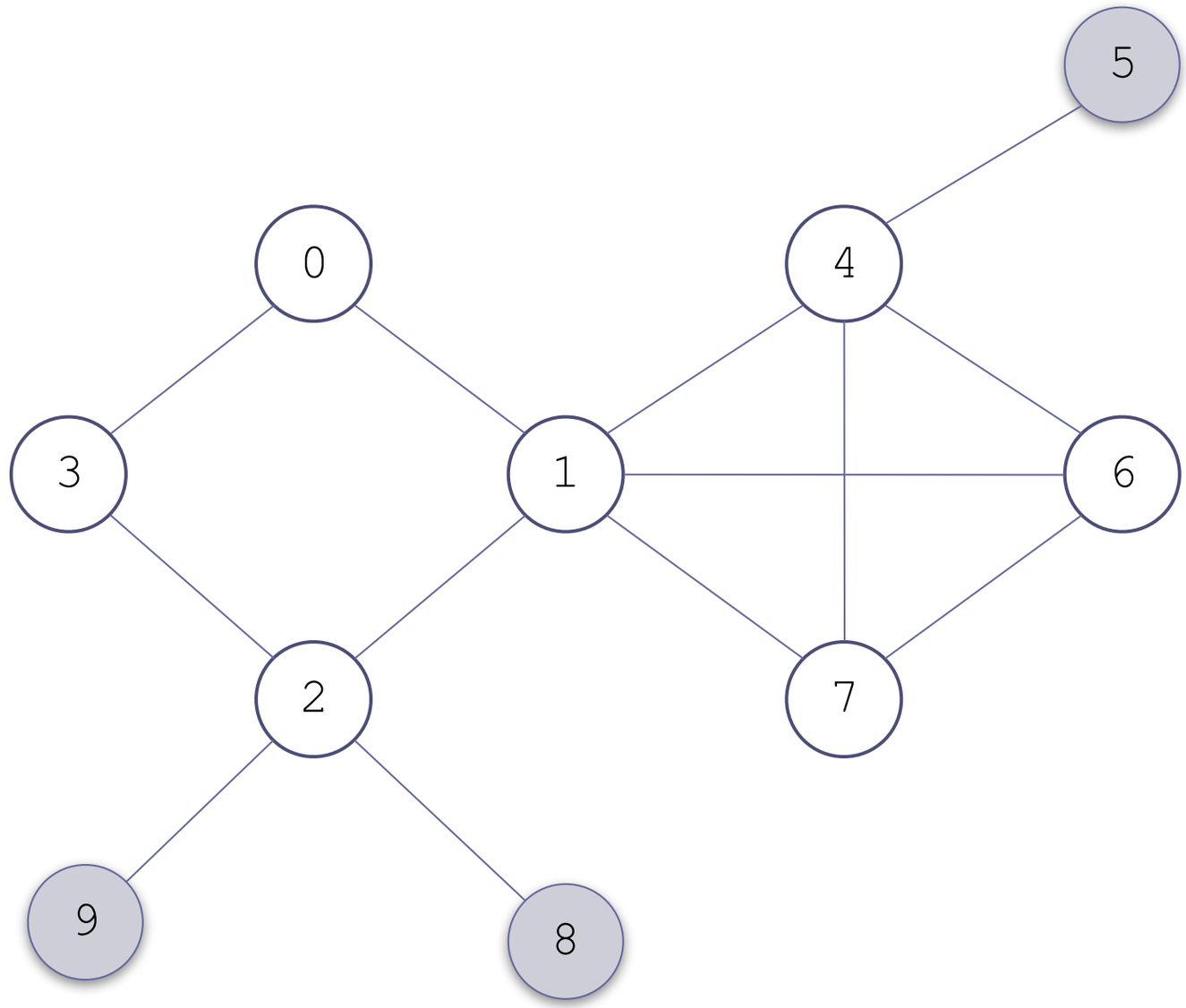
Queue:
8, 9, 5

Visit sequence:
0, 1, 3, 2, 4, 6, 7



Example of a Breadth-First Search (cont.)

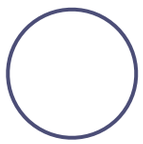
We go back to the vertices of 2 and visit them



Queue:
8, 9, 5

Visit sequence:
0, 1, 3, 2, 4, 6, 7

unvisited



visited

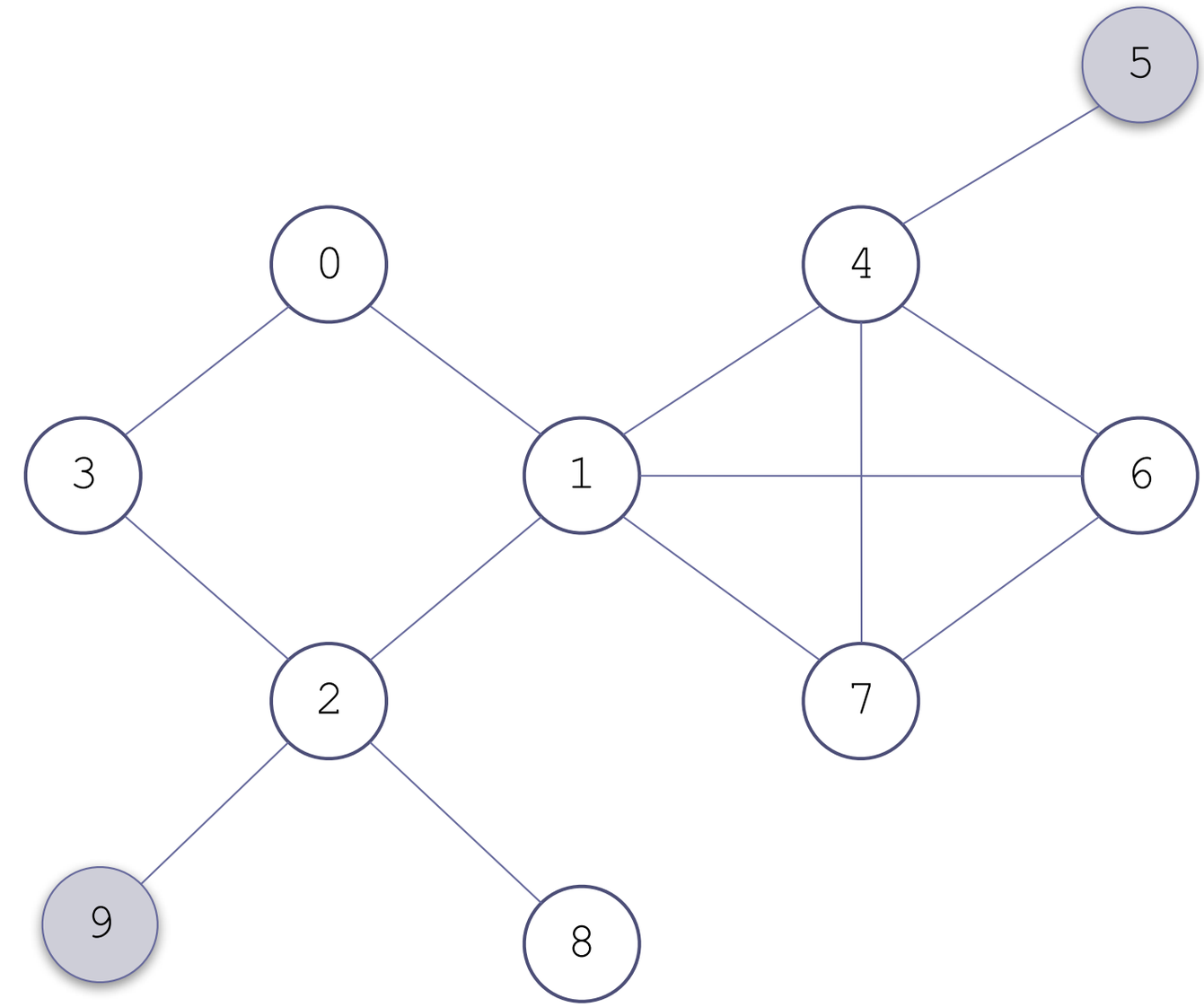
identified

Example of a Breadth-First Search (cont.)

8 has no vertices not already visited or identified

Queue:
9, 5

Visit sequence:
0, 1, 3, 2, 4, 6, 7, 8

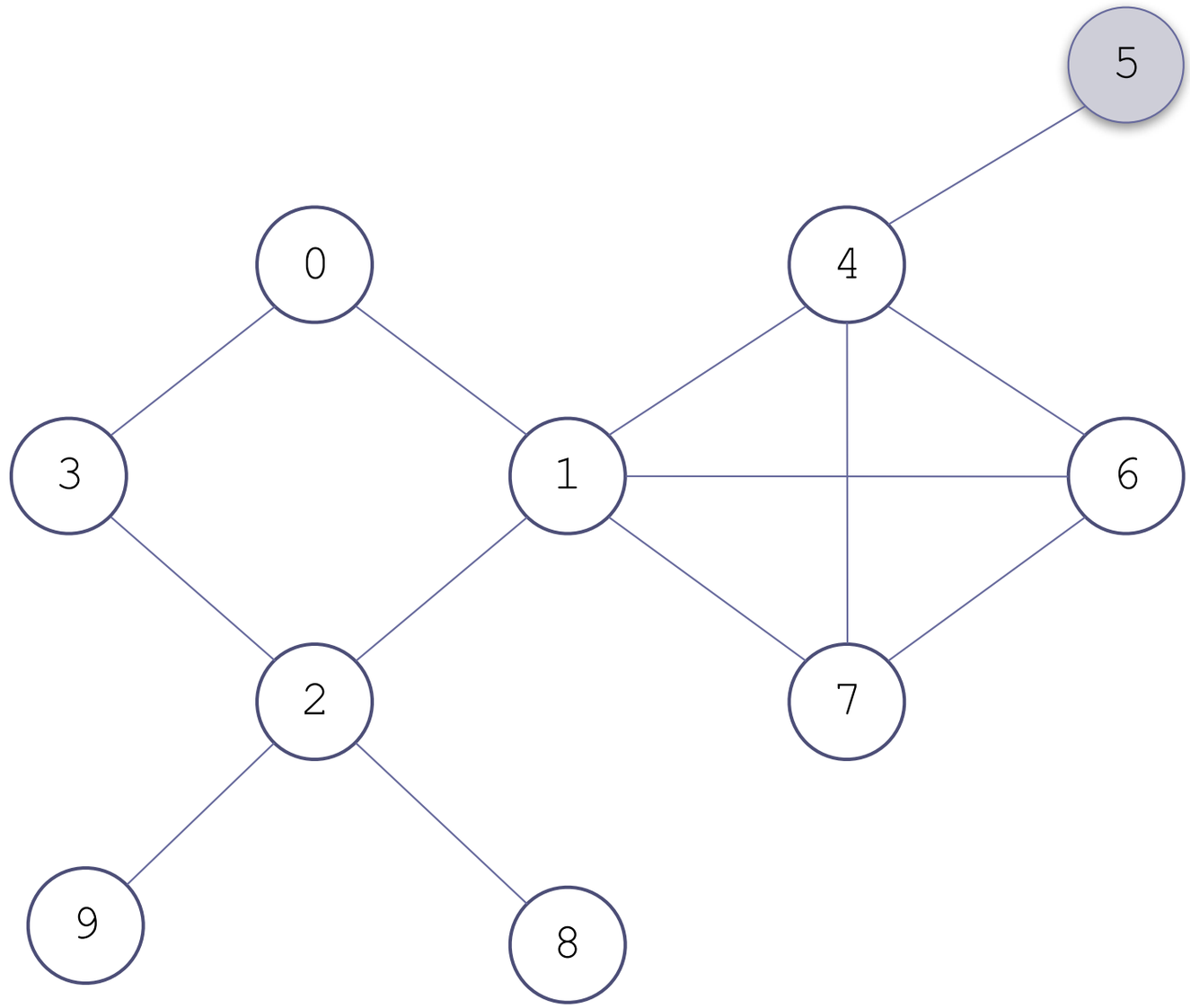


Example of a Breadth-First Search (cont.)

9 has no vertices not already visited or identified

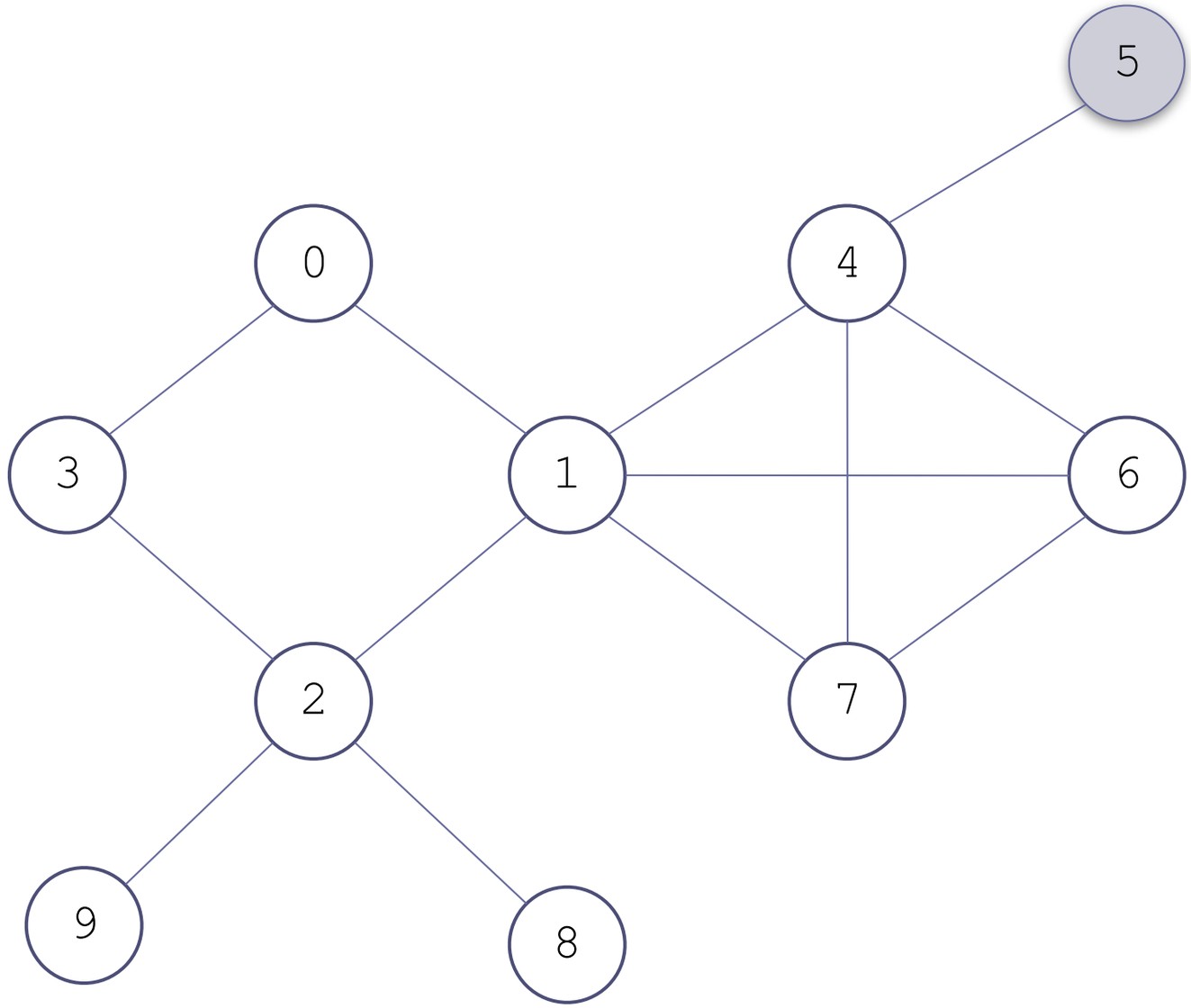
Queue:
5

Visit sequence:
0, 1, 3, 2, 4, 6, 7, 8, 9



Example of a Breadth-First Search (cont.)

Finally we visit 5



Queue:
5

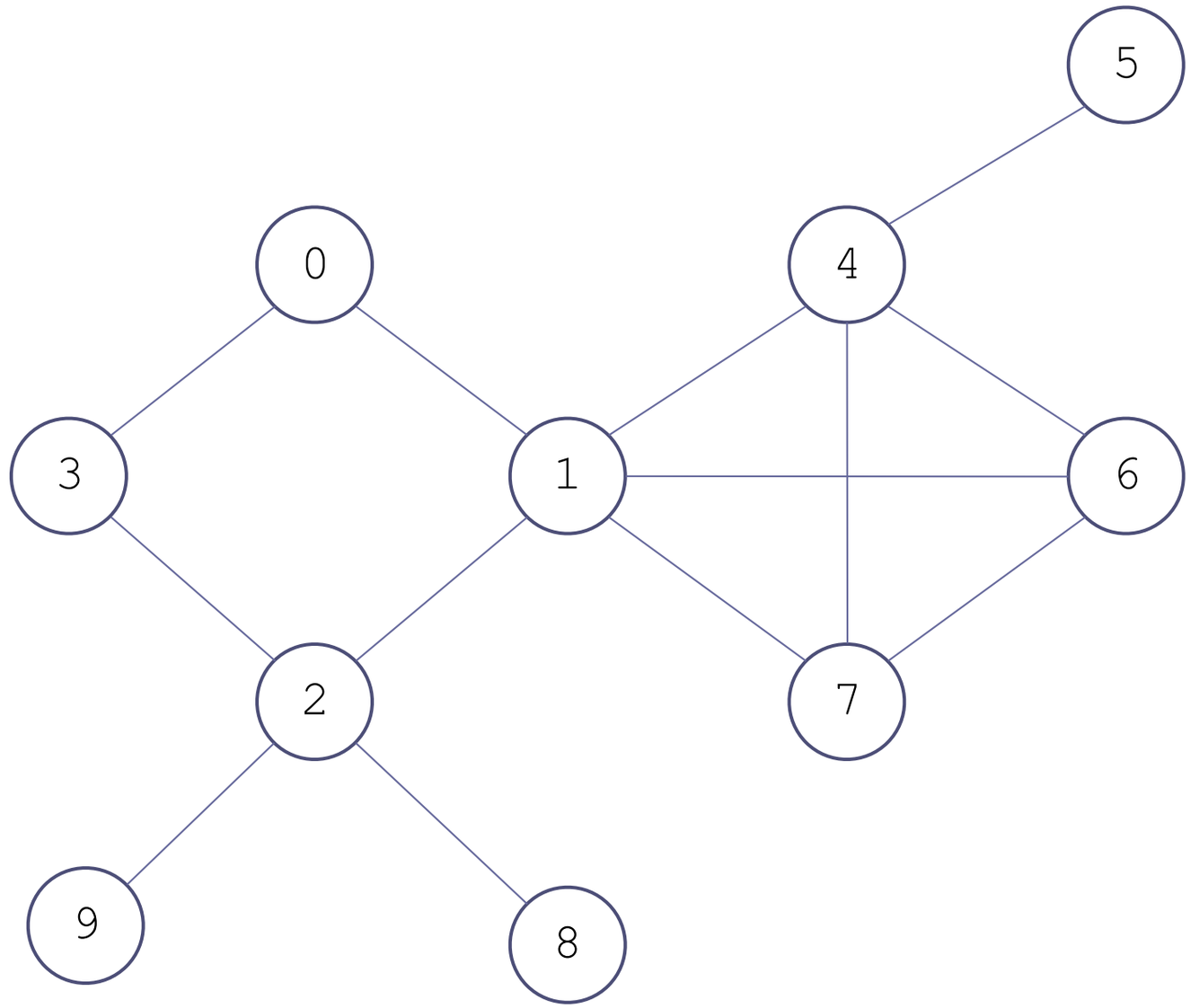
Visit sequence:
0, 1, 3, 2, 4, 6, 7, 8, 9

Example of a Breadth-First Search (cont.)

which has no vertices not already visited or identified

Queue:
empty

Visit sequence:
0, 1, 3, 2, 4, 6, 7, 8, 9, 5



0 **unvisited**

visited

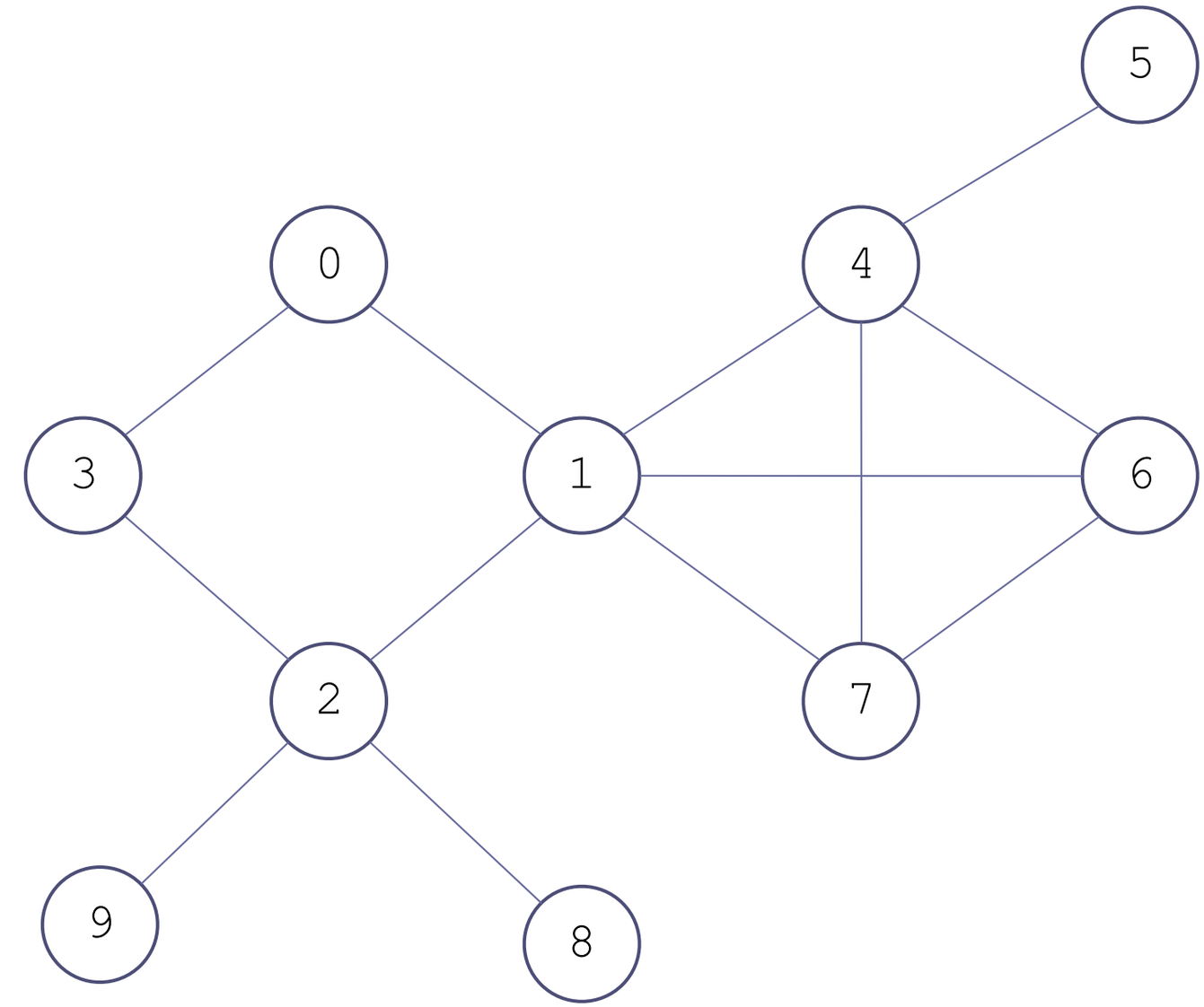
0 **identified**

Example of a Breadth-First Search (cont.)

The queue is empty; all vertices have been visited

Queue:
empty

Visit sequence:
0, 1, 3, 2, 4, 6, 7, 8, 9, 5



0 **unvisited** **visited** 0 **identified**

Algoritm för bredden-först

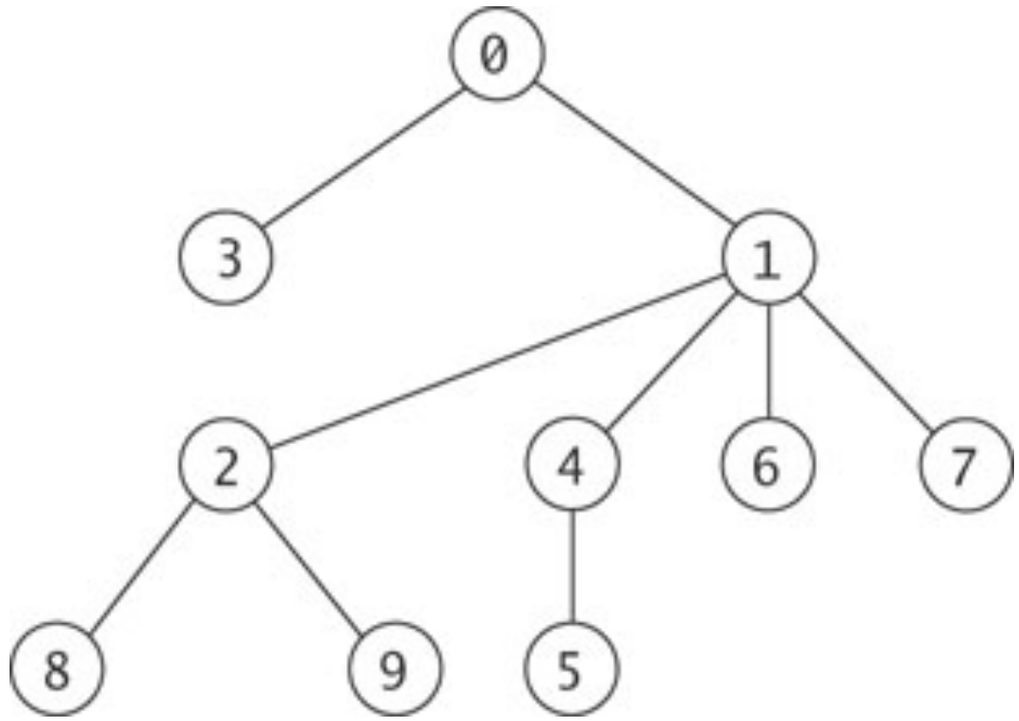
Algorithm for Breadth-First Search

1. Take an arbitrary start vertex, mark it identified (color it light blue), and place it in a queue.
2. `while` the queue is not empty
3. Take a vertex, u, out of the queue and visit u.
4. `for` all vertices, v, adjacent to this vertex, u
5. `if` v has not been identified or visited
6. Mark it identified (color it light blue).
7. Insert vertex v into the queue.
8. We are now finished visiting u (color it dark blue).

Bredden-först-sökträdet

Vi kan bygga ett träd som består av de bågar som vi faktiskt utnyttjade vid sökningen.

Detta träd har alla noder och en del av bågarna från originalgrafen.



Informationen som behövs för att representera sökträdet, kan lagras i ett fält: Där lagrar vi föräldern till varje båge när den identifieras.

Vi kan förfina steg 7 i algoritmen såhär:

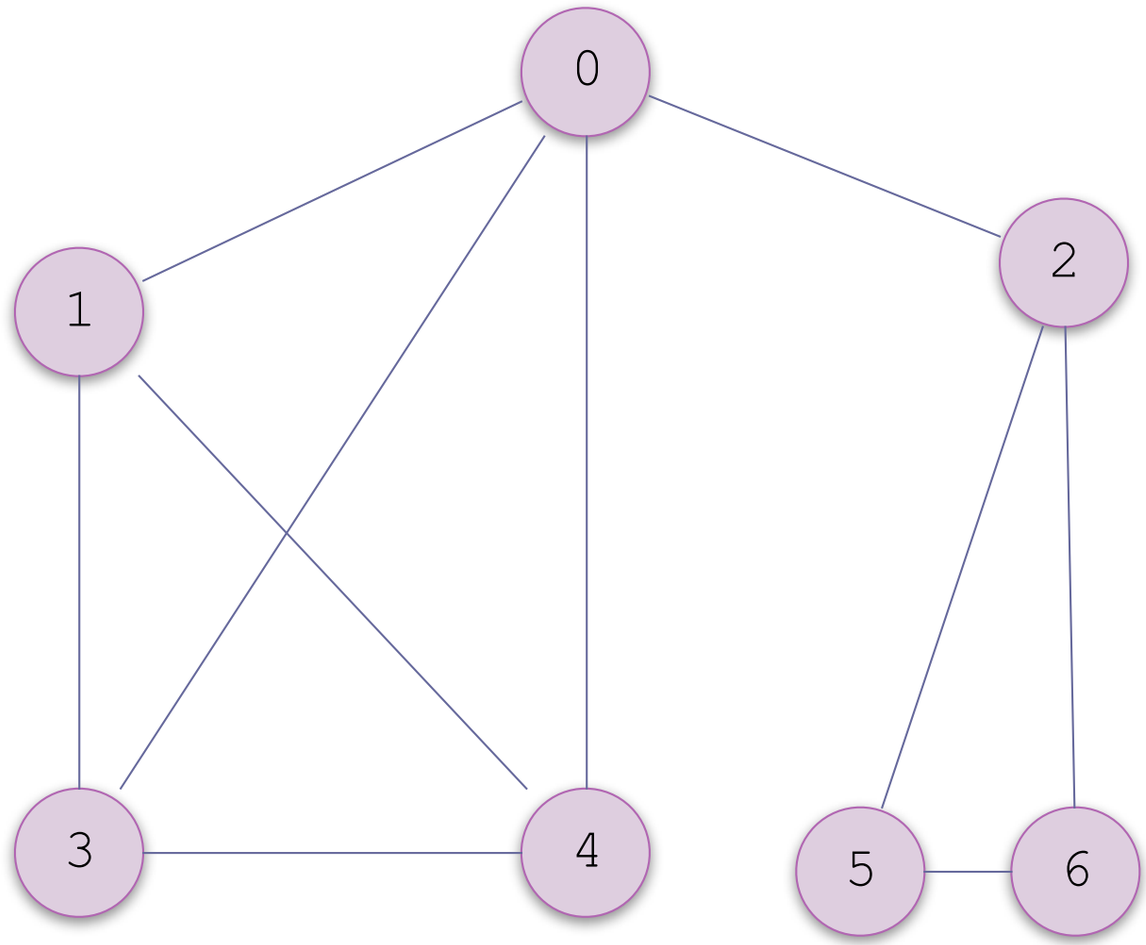
- 7.1. Stoppa in nod *v* i kön
- 7.2. Sätt *v*:s förälder till *u*

DFS: Djupet-först-sökning

Vid djupet-först-sökning så besöker vi noderna i följande ordning:

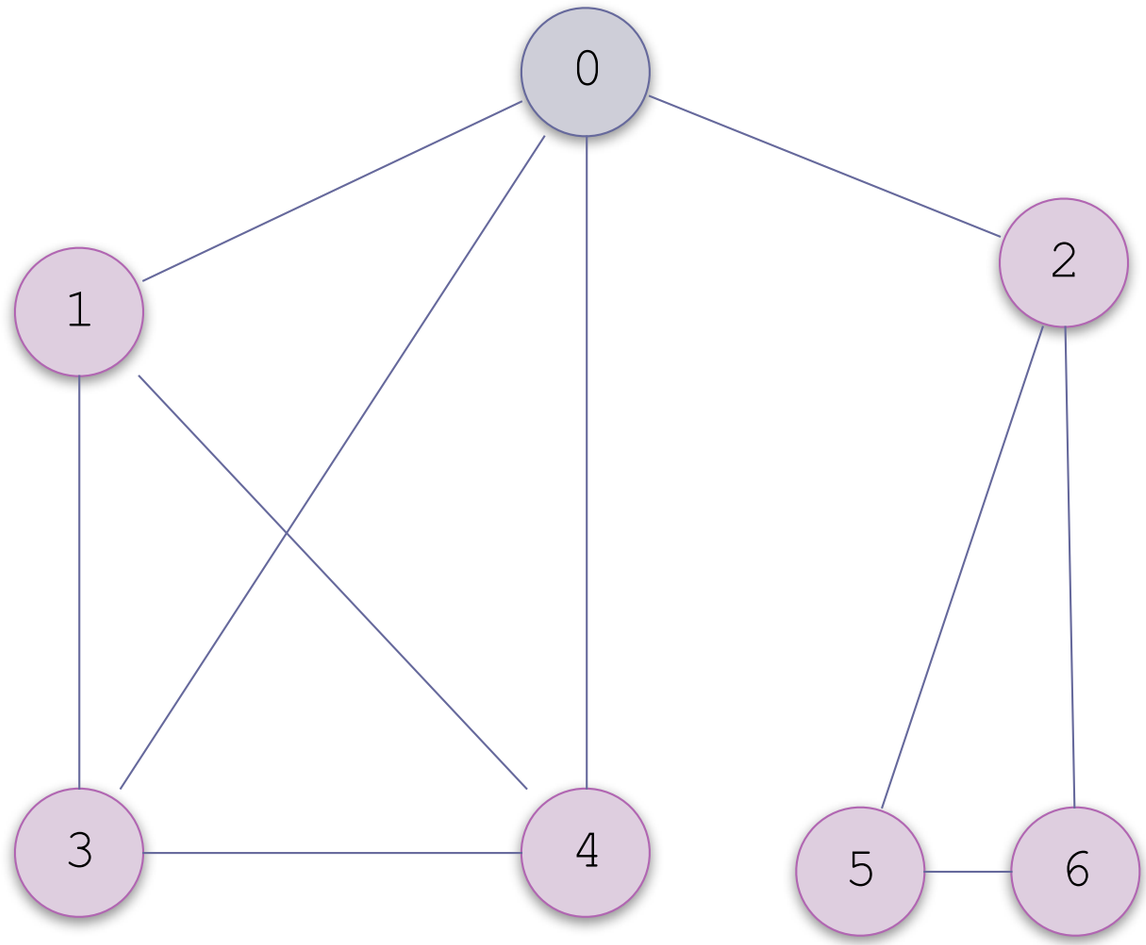
- besök startnoden först
- välj en angränsande nod att besöka
- sedan en angränsande nod till denna
- och så vidare tills det inte finns några fler noder
- sedan backar vi och kollar ifall vi kan hitta en annan angränsande båge
- och så vidare

Example of a Depth-First Search



Example of a Depth-First Search (cont.)

Mark 0 as being visited



Discovery (Visit) order:
0

Finish order:

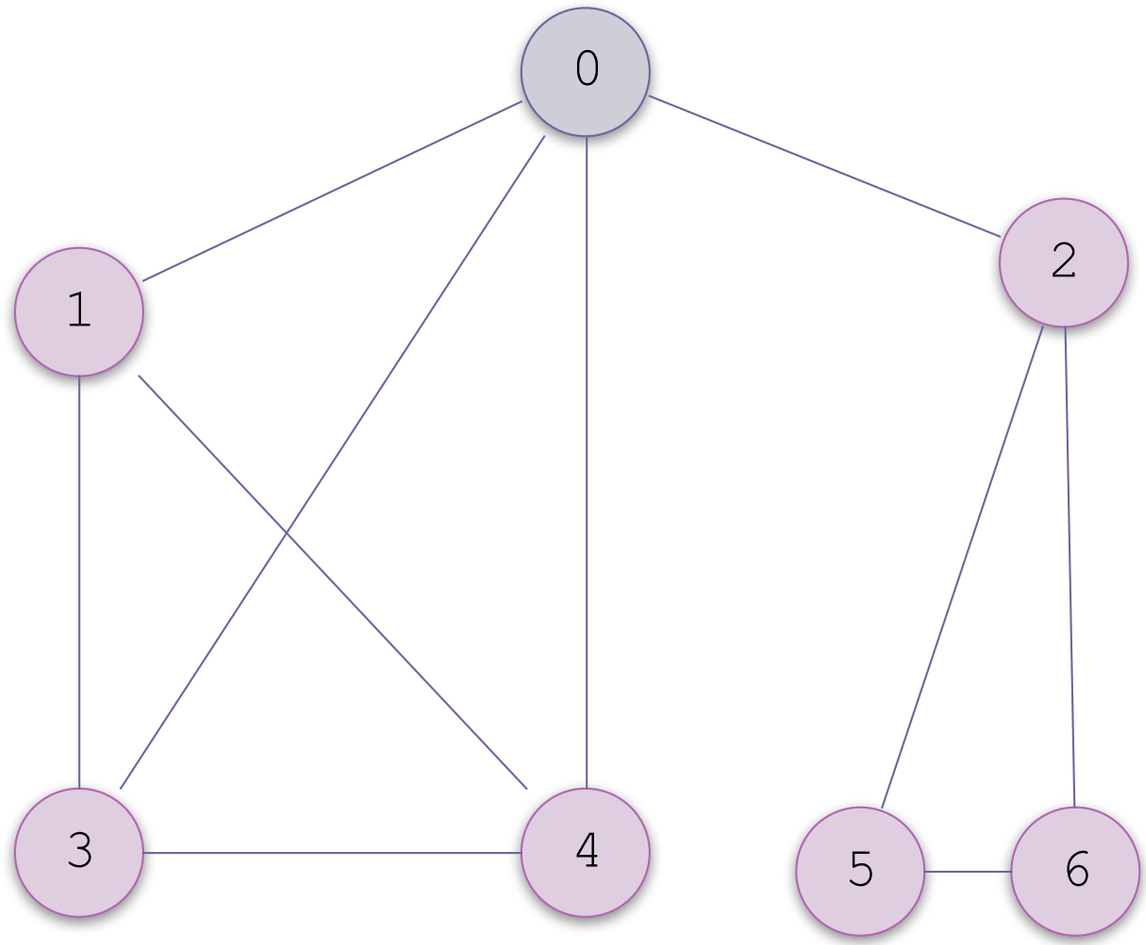
0 **unvisited**

visited

0 **being visited**

Example of a Depth-First Search (cont.)

Choose an adjacent vertex that is not being visited

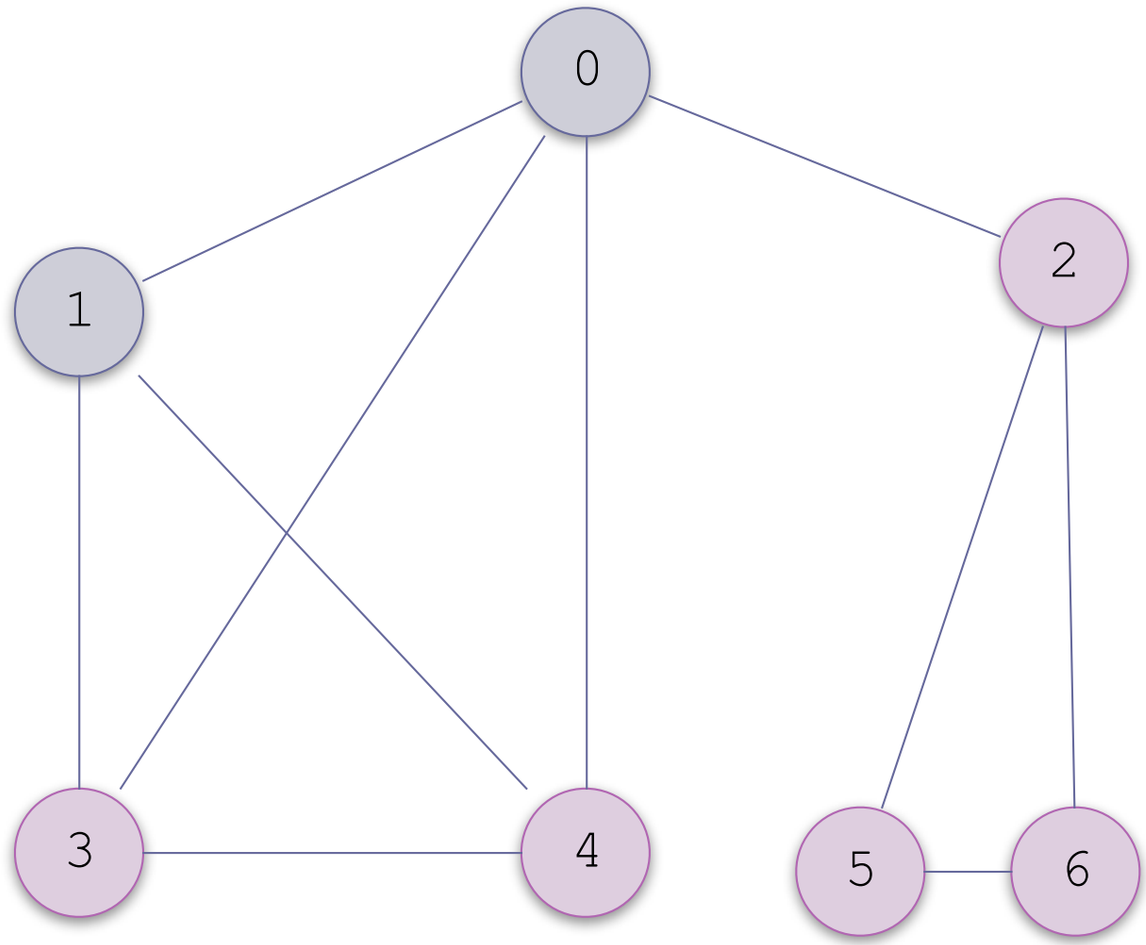


Discovery (Visit) order:
0

Finish order:

Example of a Depth-First Search (cont.)

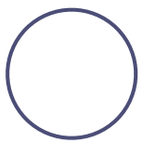
Choose an adjacent vertex that is not being visited



Discovery (Visit) order:
0, 1

Finish order:

unvisited

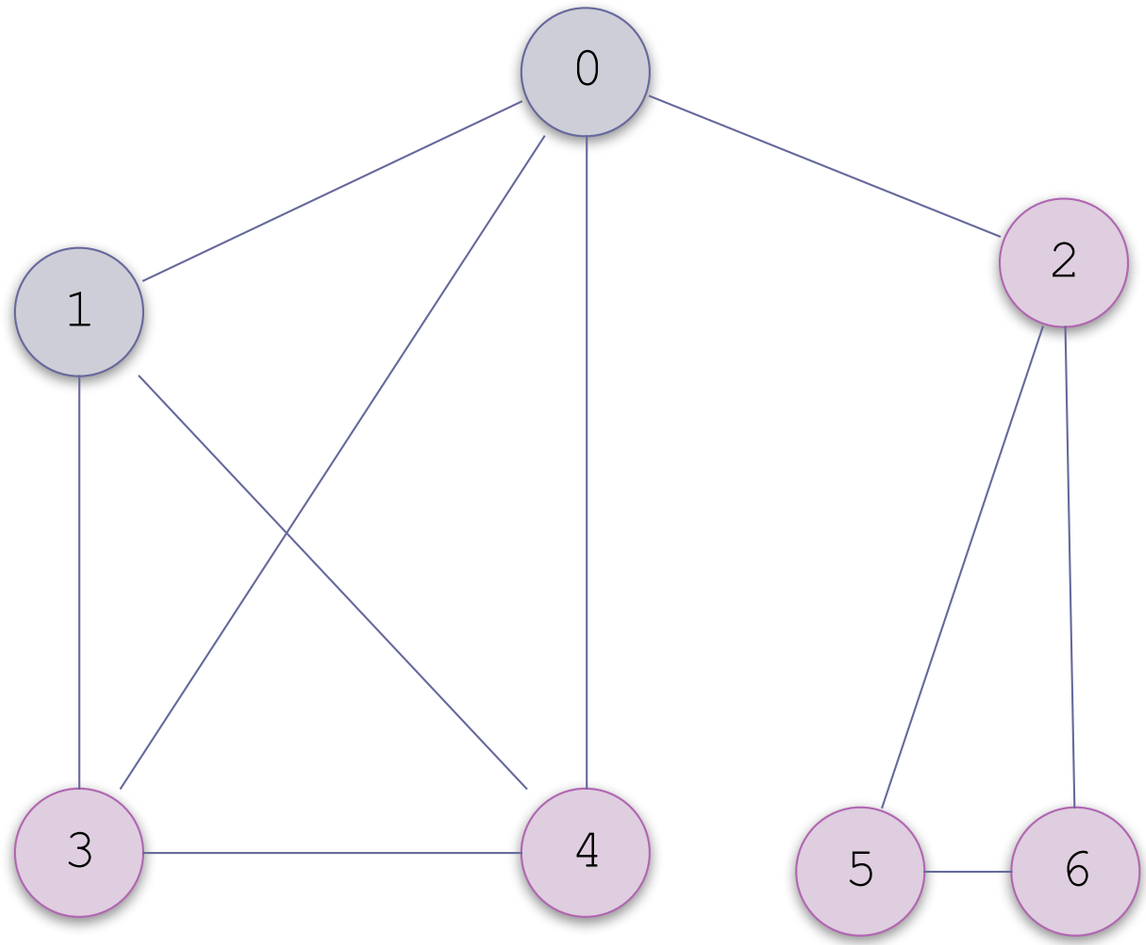


visited

being visited

Example of a Depth-First Search (cont.)

(Recursively) choose an adjacent vertex that is not being visited



Discovery (Visit) order:
0, 1, 3

Finish order:

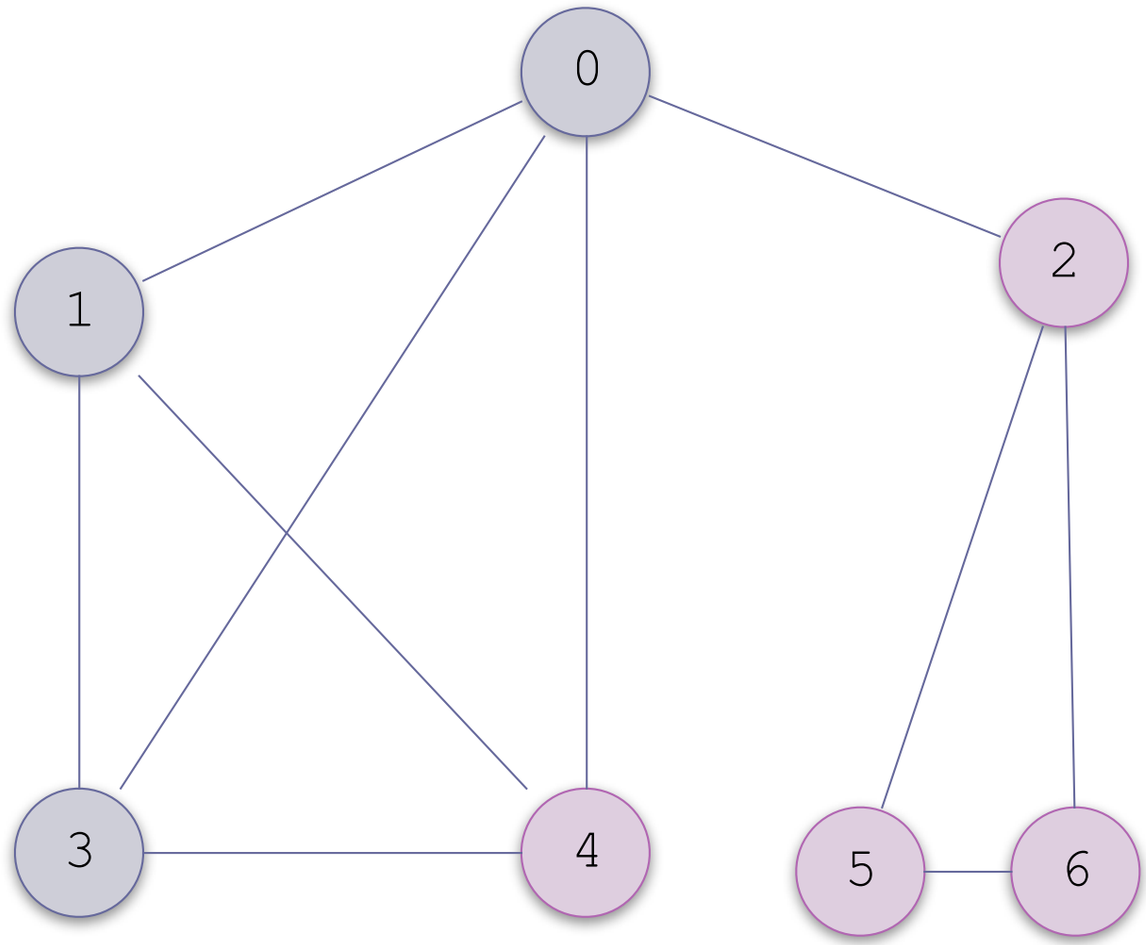
0 **unvisited**

visited

0 **being visited**

Example of a Depth-First Search (cont.)

(Recursively) choose an adjacent vertex that is not being visited



Discovery (Visit) order:
0, 1, 3

Finish order:

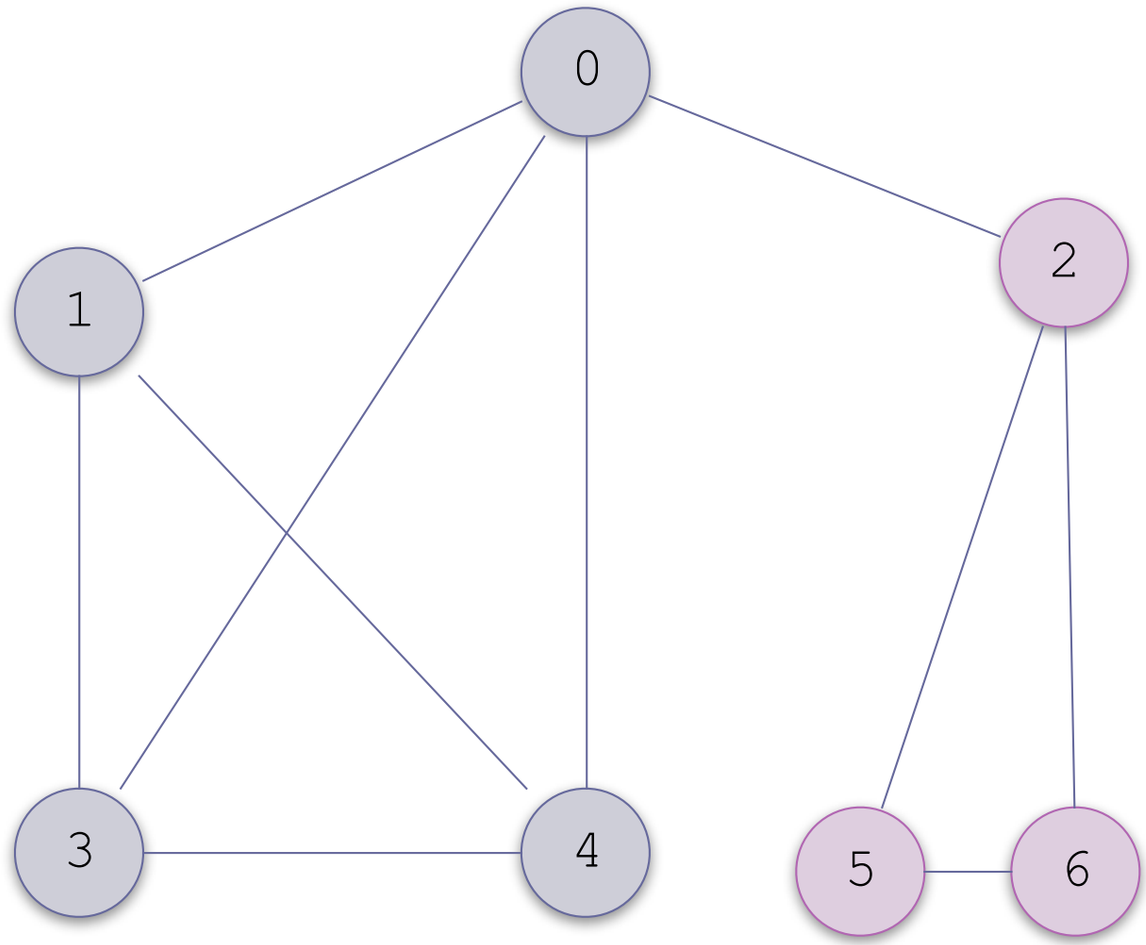
0 **unvisited**

visited

0 **being visited**

Example of a Depth-First Search (cont.)

(Recursively) choose an adjacent vertex that is not being visited



Discovery (Visit) order:
0, 1, 3, 4

Finish order:

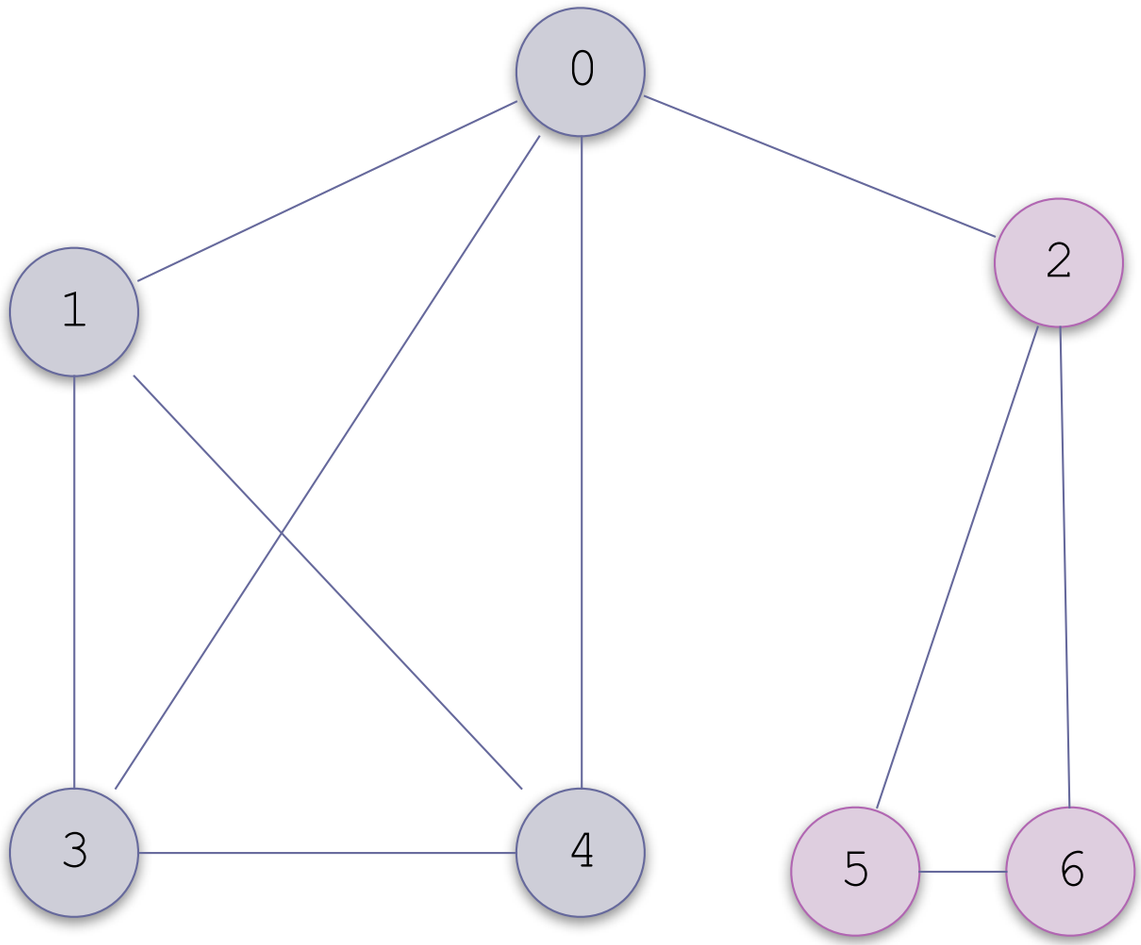
0 **unvisited**

visited

0 **being visited**

Example of a Depth-First Search (cont.)

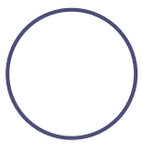
There are no vertices adjacent to 4 that are not being visited



Discovery (Visit) order:
0, 1, 3, 4

Finish order:

unvisited

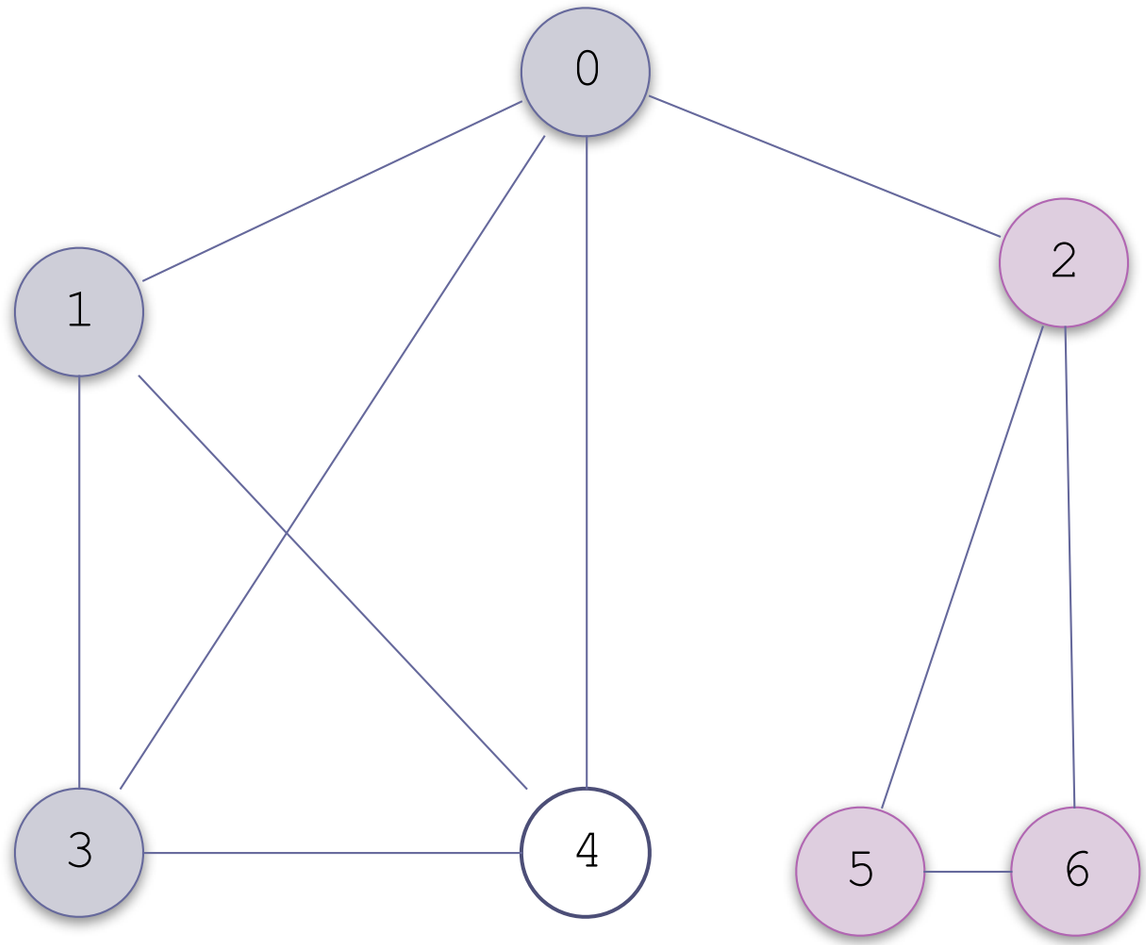


visited

being visited

Example of a Depth-First Search (cont.)

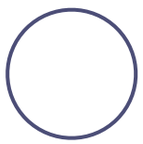
Mark 4 as visited



Discovery (Visit) order:
0, 1, 3, 4

Finish order:
4

unvisited

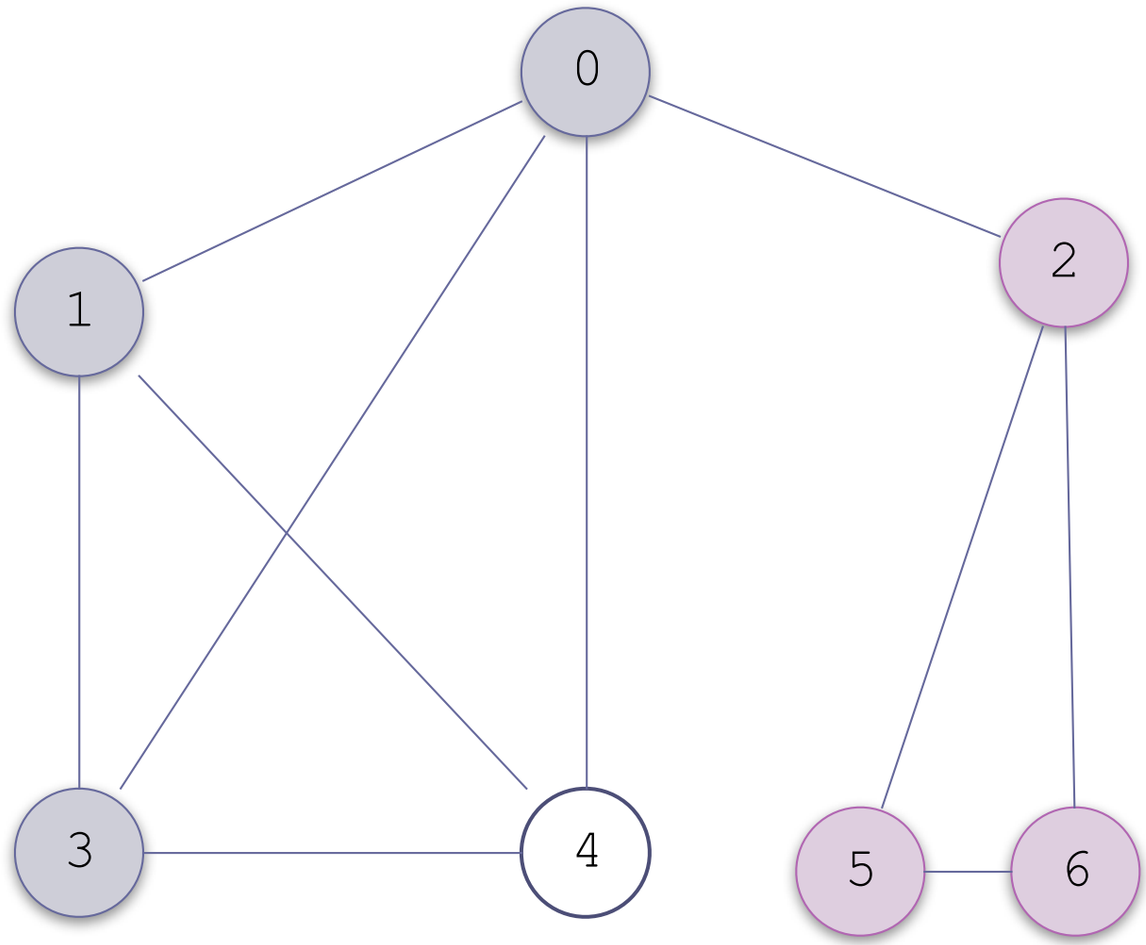


visited

being visited

Example of a Depth-First Search (cont.)

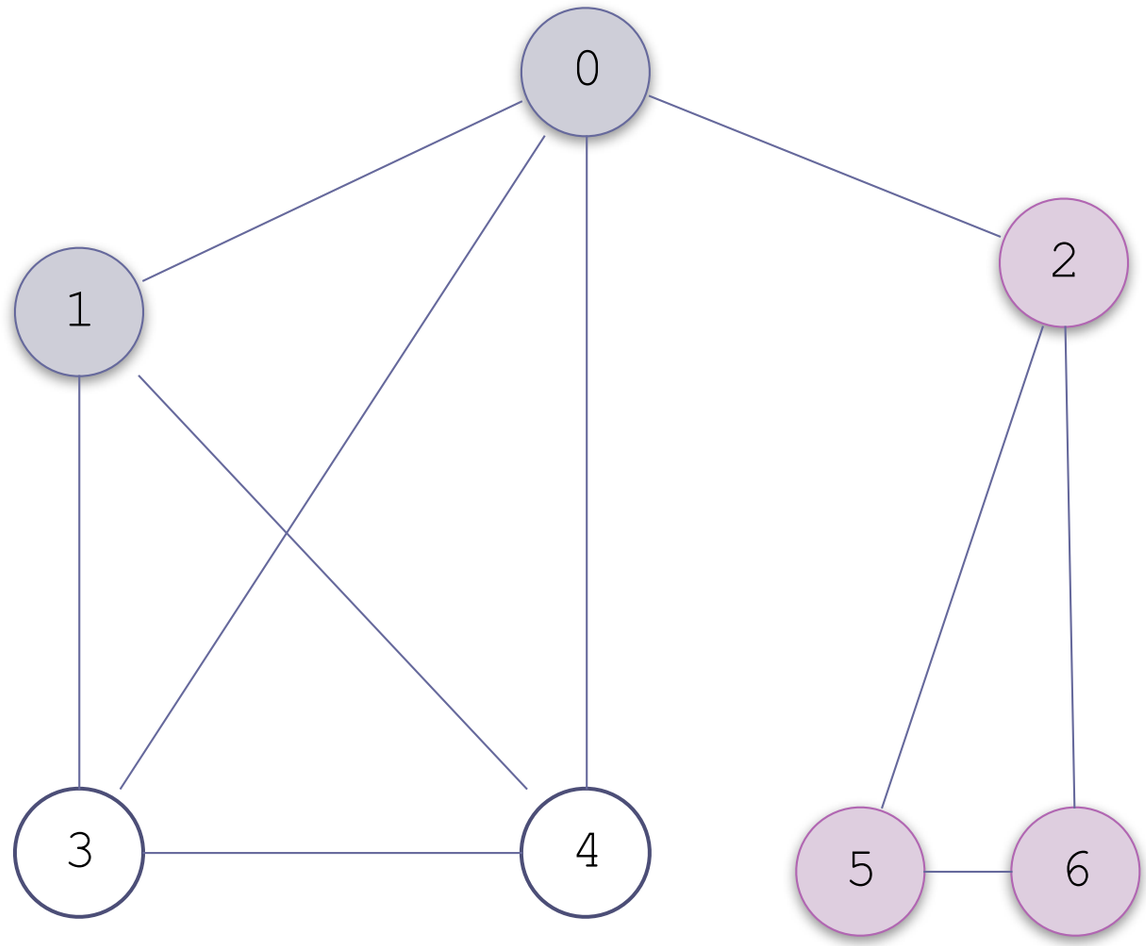
Return from the recursion to 3; all adjacent nodes to 3 are being visited



Finish order:
4

Example of a Depth-First Search (cont.)

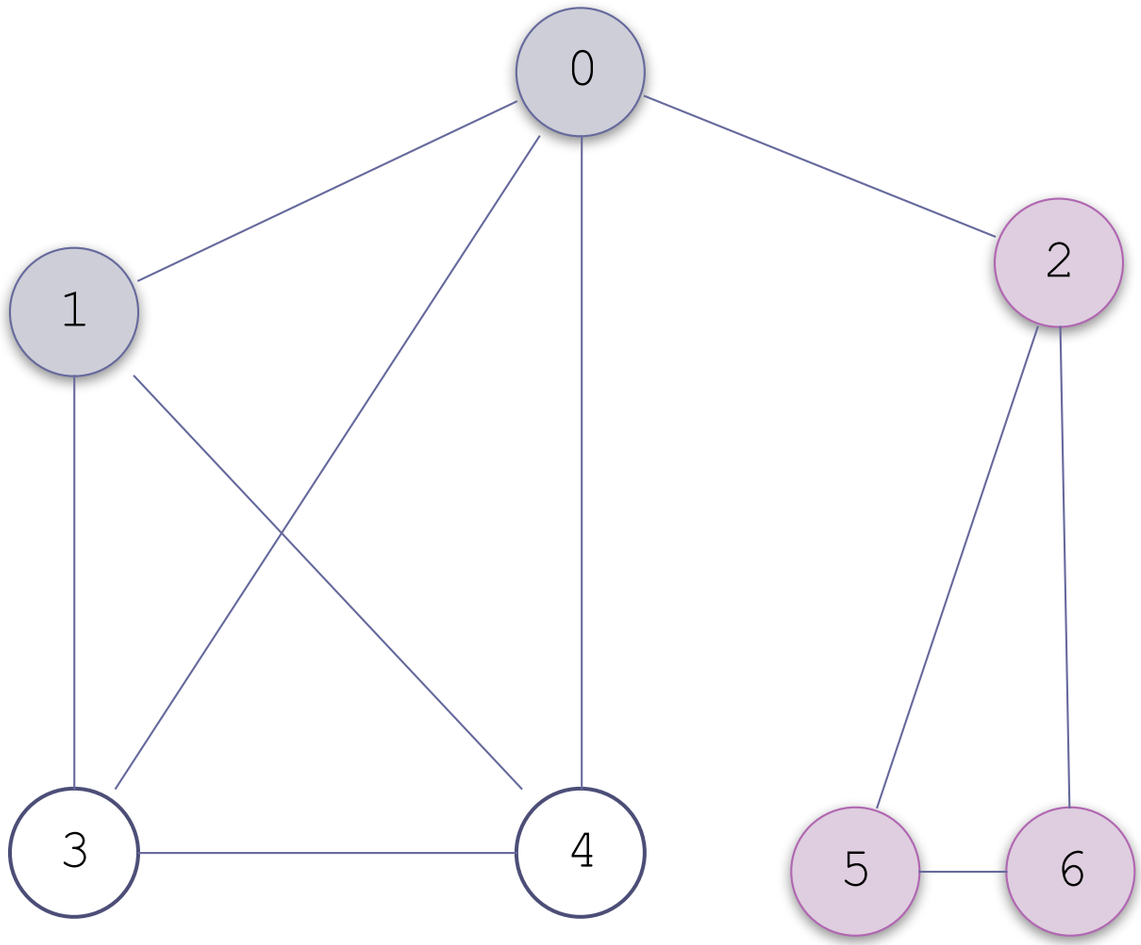
Mark 3 as visited



Finish order:
4, 3

Example of a Depth-First Search (cont.)

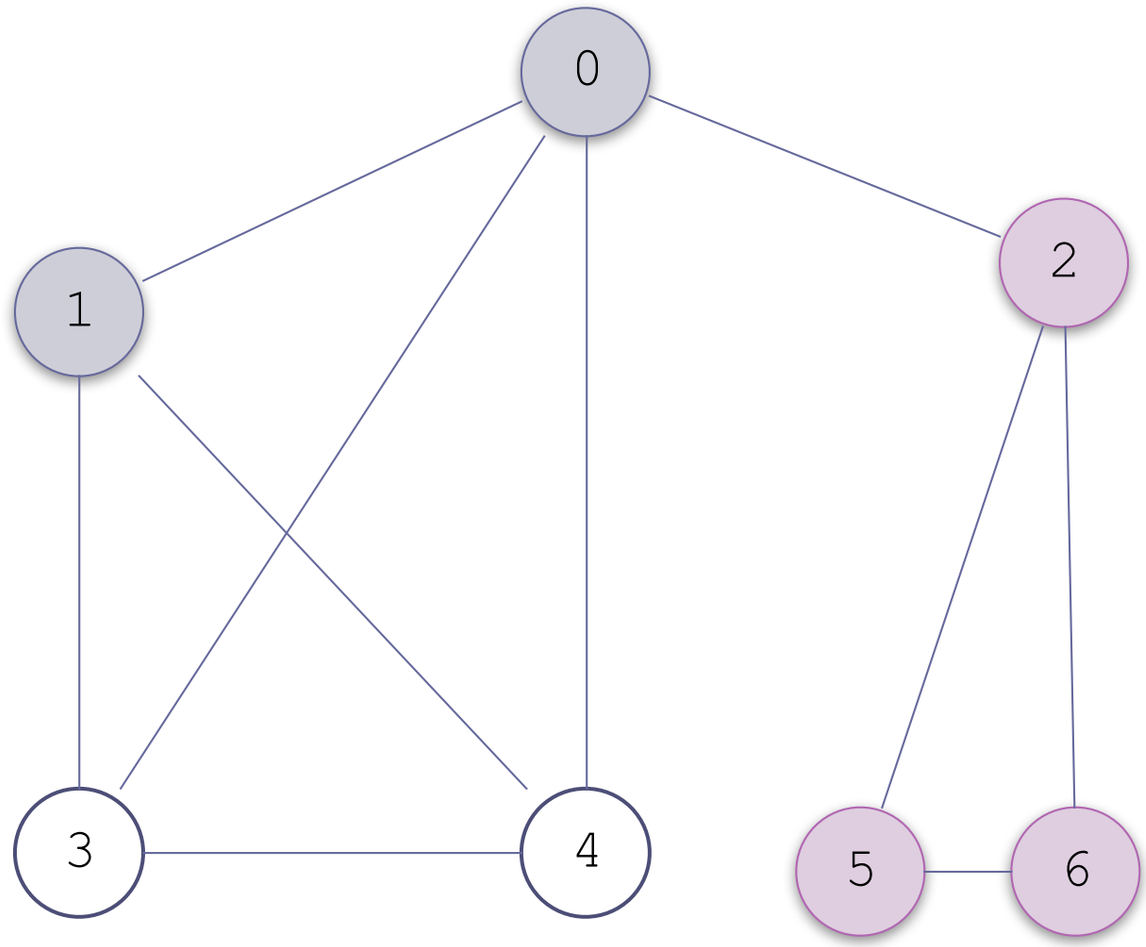
Return from the recursion to 1



Finish order:
4, 3

Example of a Depth-First Search (cont.)

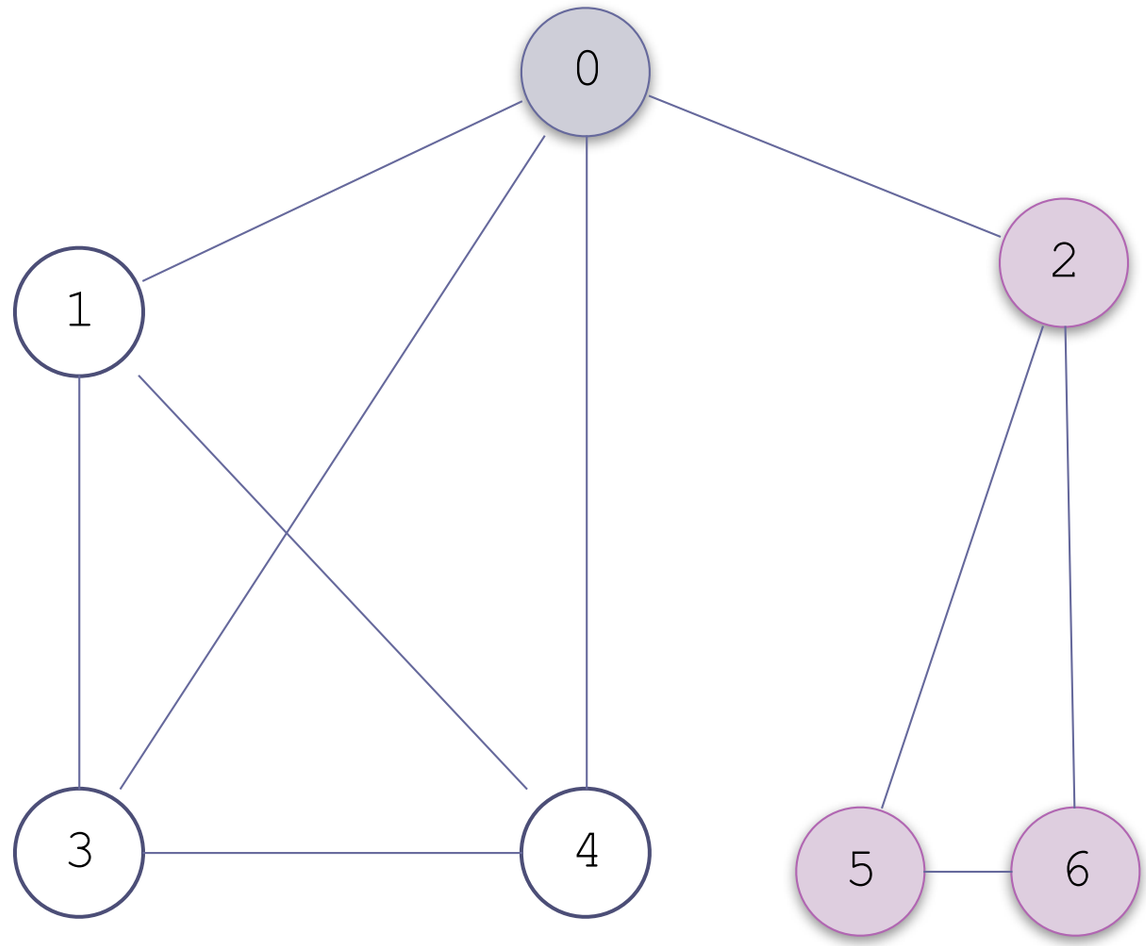
All vertices adjacent to 1 are being visited



Finish order:
4, 3

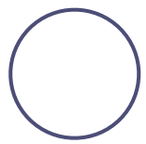
Example of a Depth-First Search (cont.)

Mark 1 as visited



Finish order:
4, 3, 1

unvisited

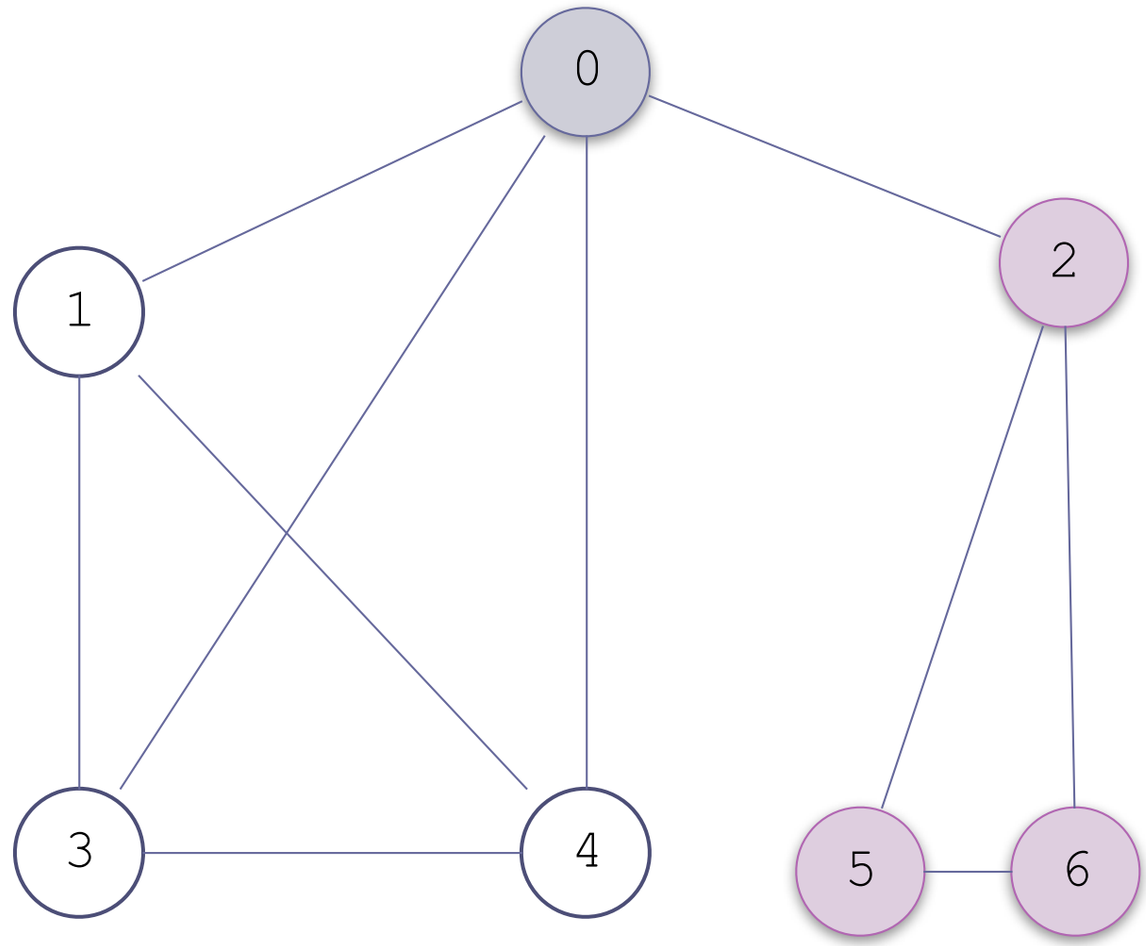


visited

being visited

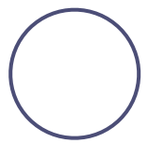
Example of a Depth-First Search (cont.)

Return from the recursion to 0



Finish order:
4, 3, 1

unvisited

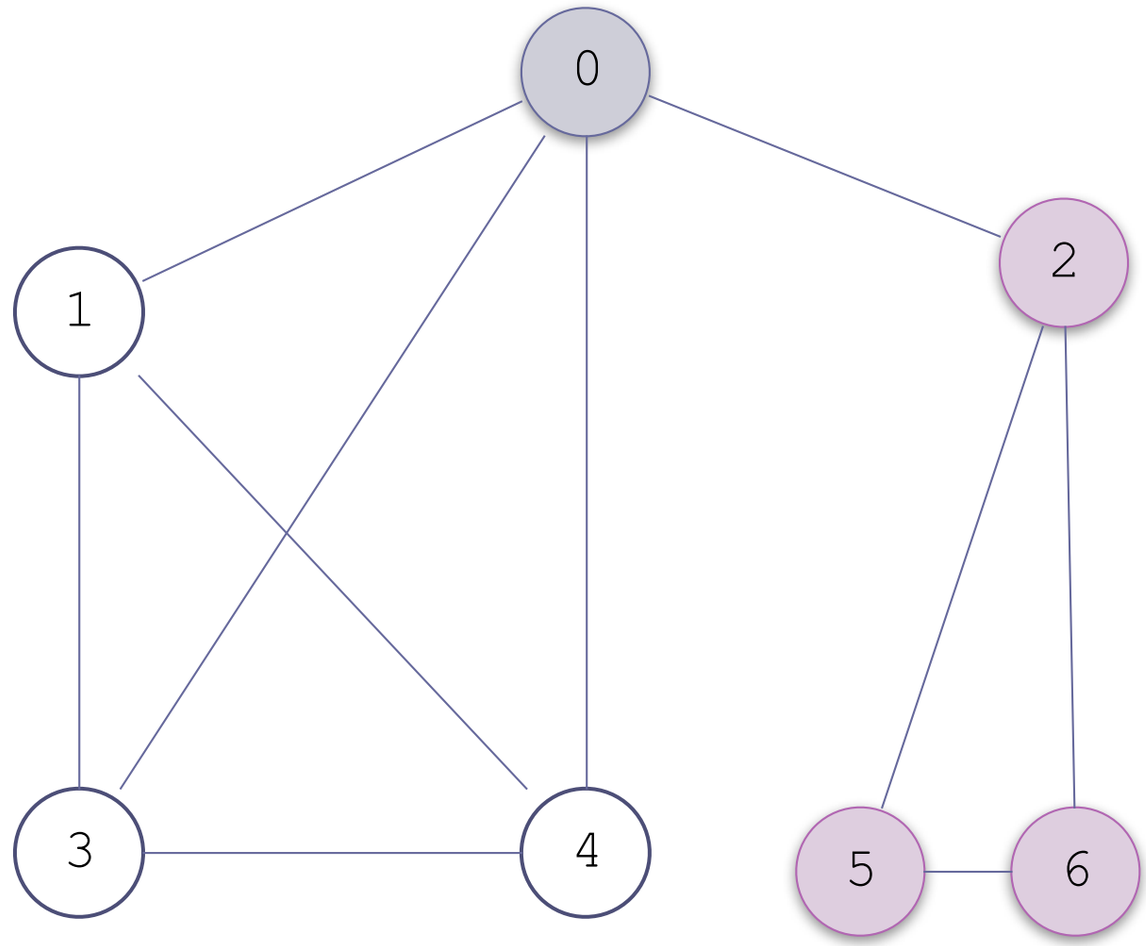


visited

being visited

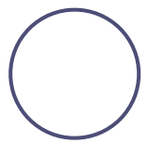
Example of a Depth-First Search (cont.)

2 is adjacent to 1 and is not being visited



Finish order:
4, 3, 1

unvisited

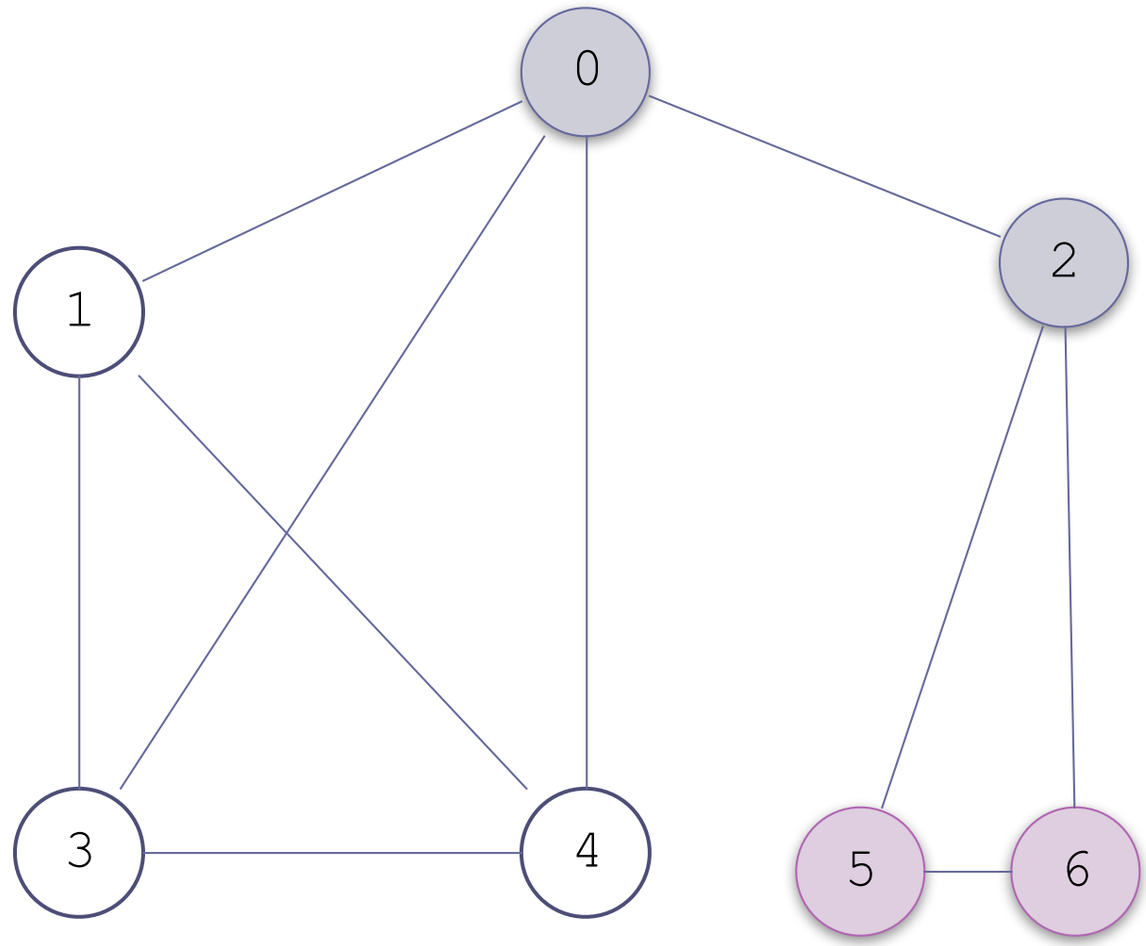


visited

being visited

Example of a Depth-First Search (cont.)

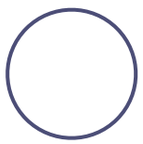
2 is adjacent to 1 and is not being visited



Discovery (Visit) order:
0, 1, 3, 4, 2

Finish order:
4, 3, 1

unvisited

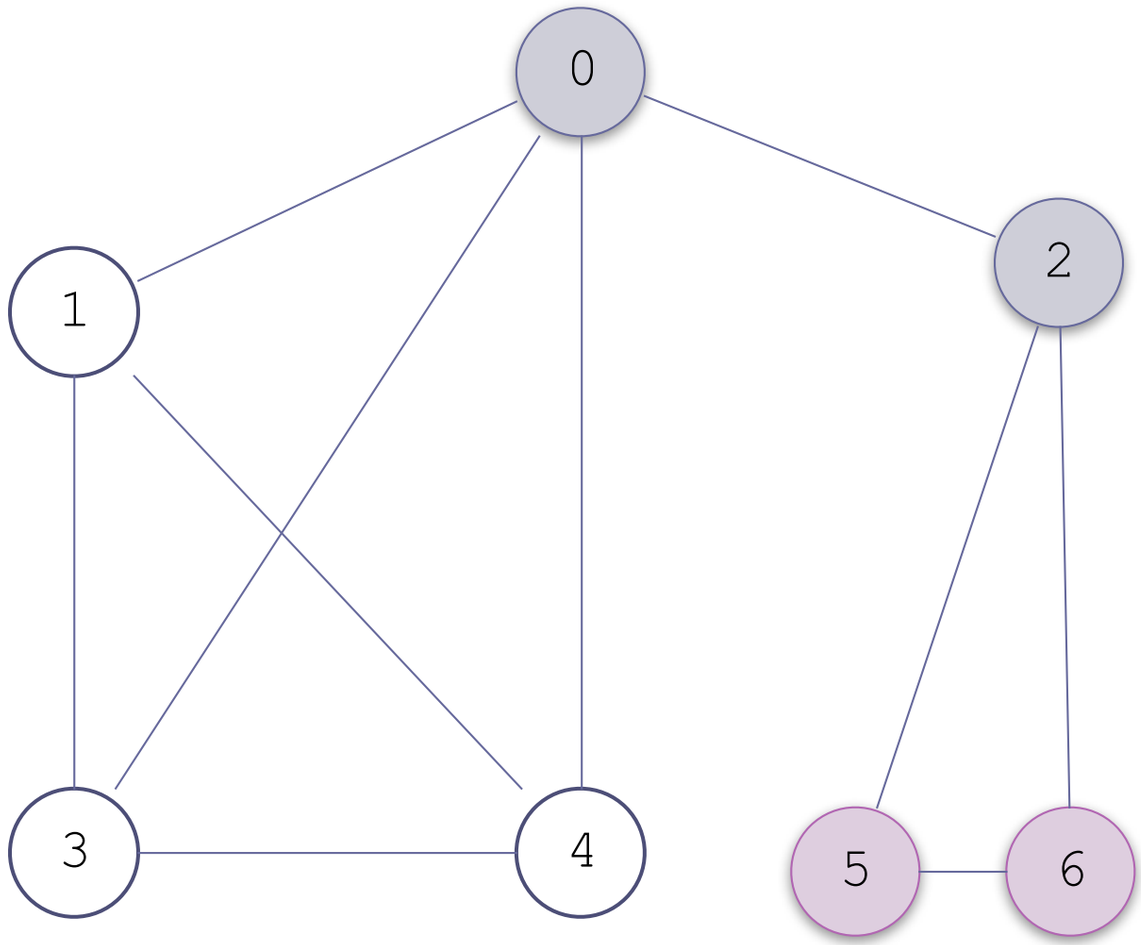


visited

being visited

Example of a Depth-First Search (cont.)

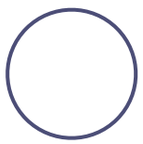
5 is adjacent to 2 and is not being visited



Discovery (Visit) order:
0, 1, 3, 4, 2

Finish order:
4, 3, 1

unvisited

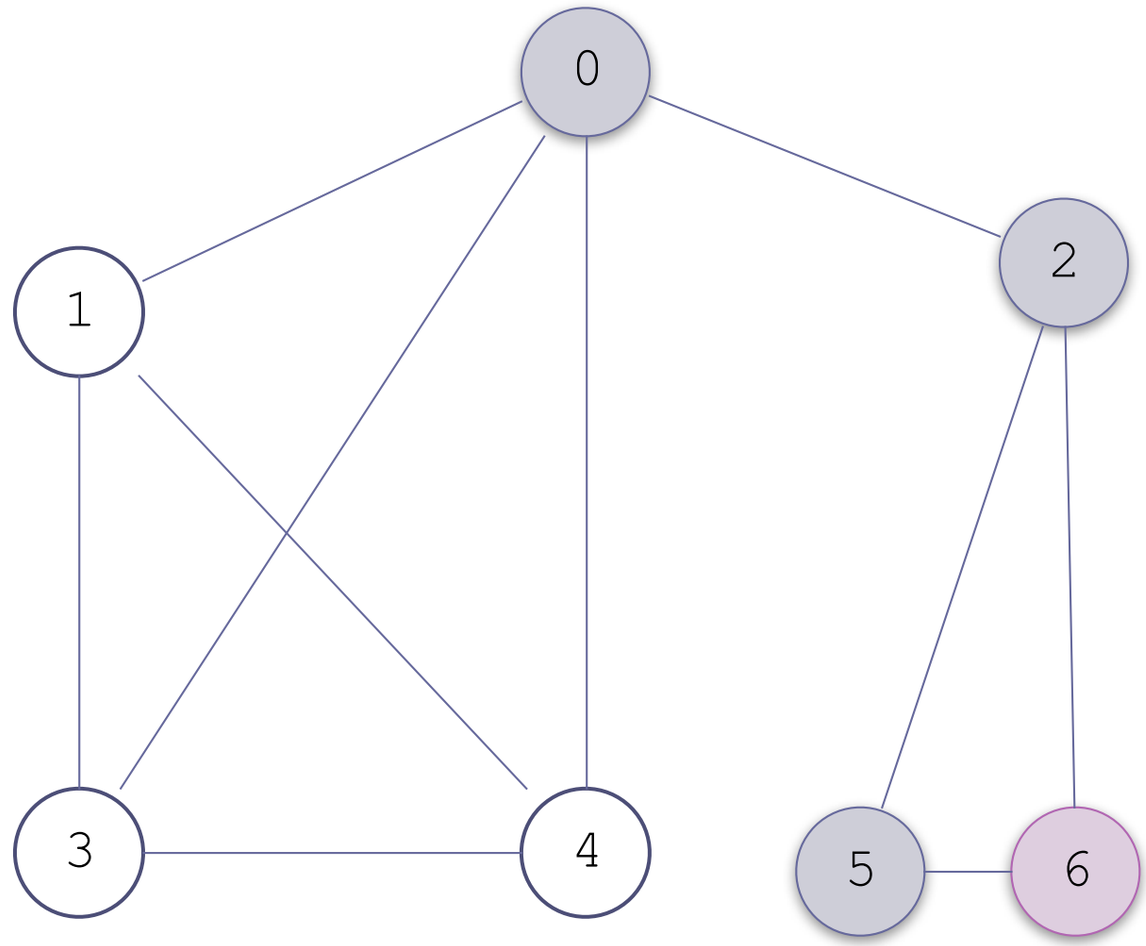


visited

being visited

Example of a Depth-First Search (cont.)

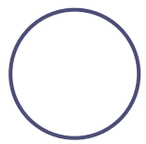
5 is adjacent to 2 and is not being visited



Discovery (Visit) order:
0, 1, 3, 4, 2, 5

Finish order:
4, 3, 1

unvisited

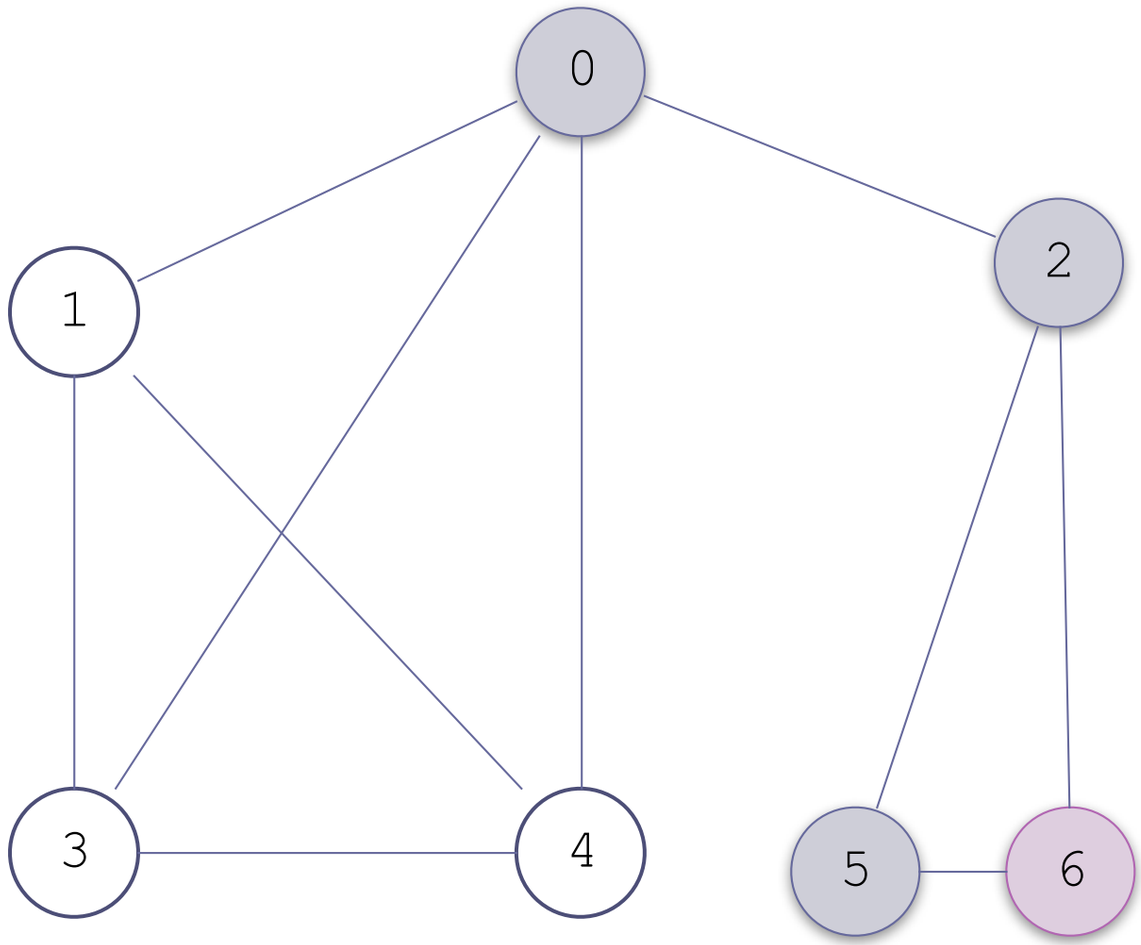


visited

being visited

Example of a Depth-First Search (cont.)

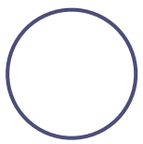
6 is adjacent to 5 and is not being visited



Discovery (Visit) order:
0, 1, 3, 4, 2, 5

Finish order:
4, 3, 1

unvisited

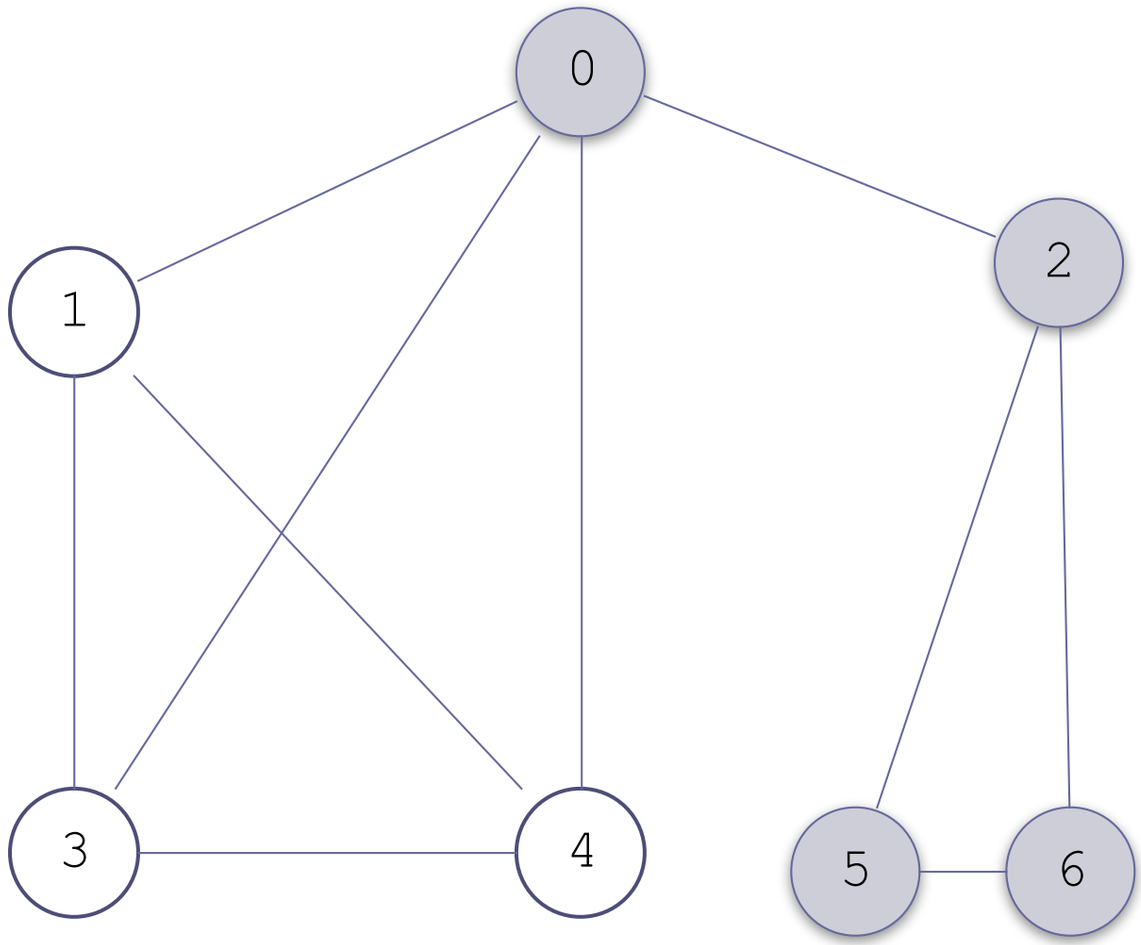


visited

being visited

Example of a Depth-First Search (cont.)

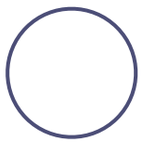
6 is adjacent to 5 and is not being visited



Discovery (Visit) order:
0, 1, 3, 4, 2, 5, 6

Finish order:
4, 3, 1

unvisited

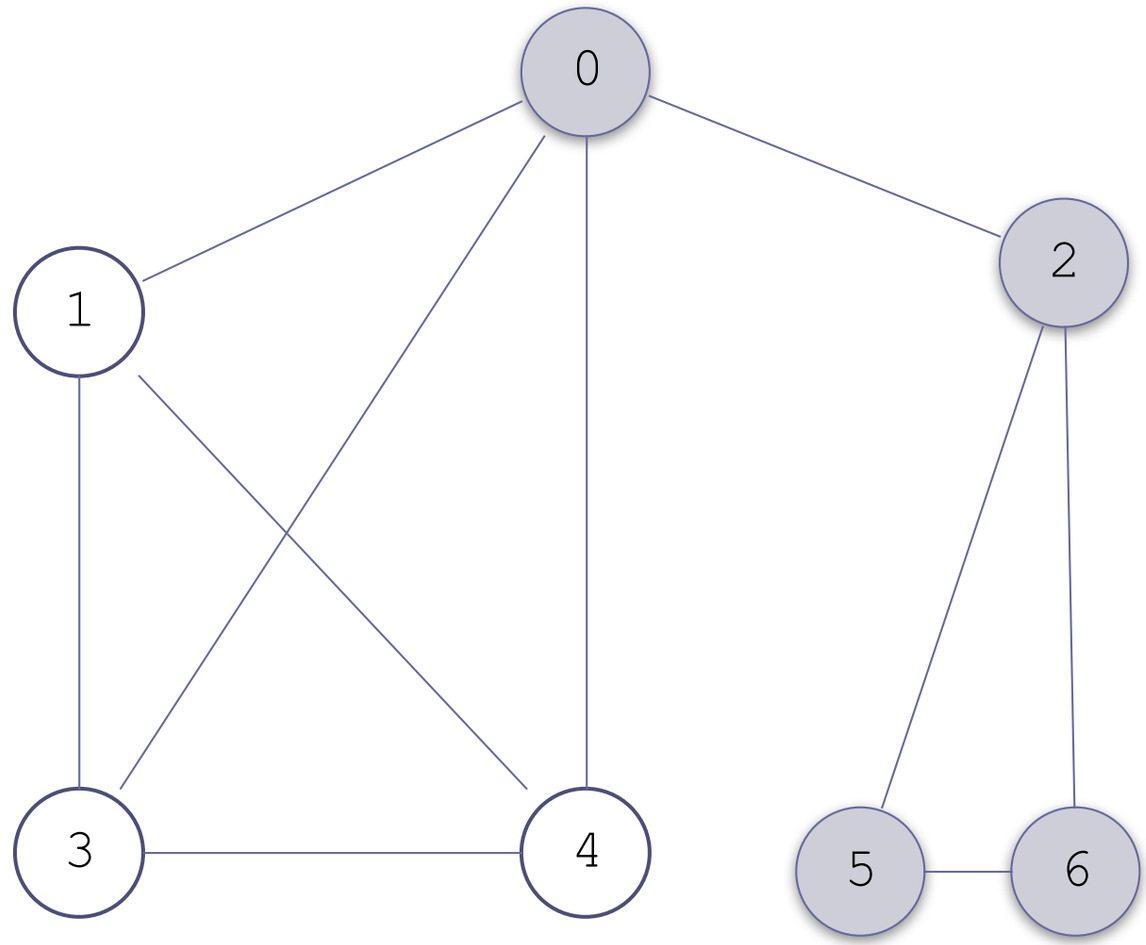


visited

being visited

Example of a Depth-First Search (cont.)

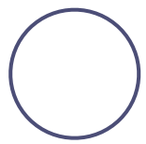
There are no vertices adjacent to 6 not being visited; mark 6 as visited



Discovery (Visit) order:
0, 1, 3, 4, 2, 5, 6

Finish order:
4, 3, 1

unvisited

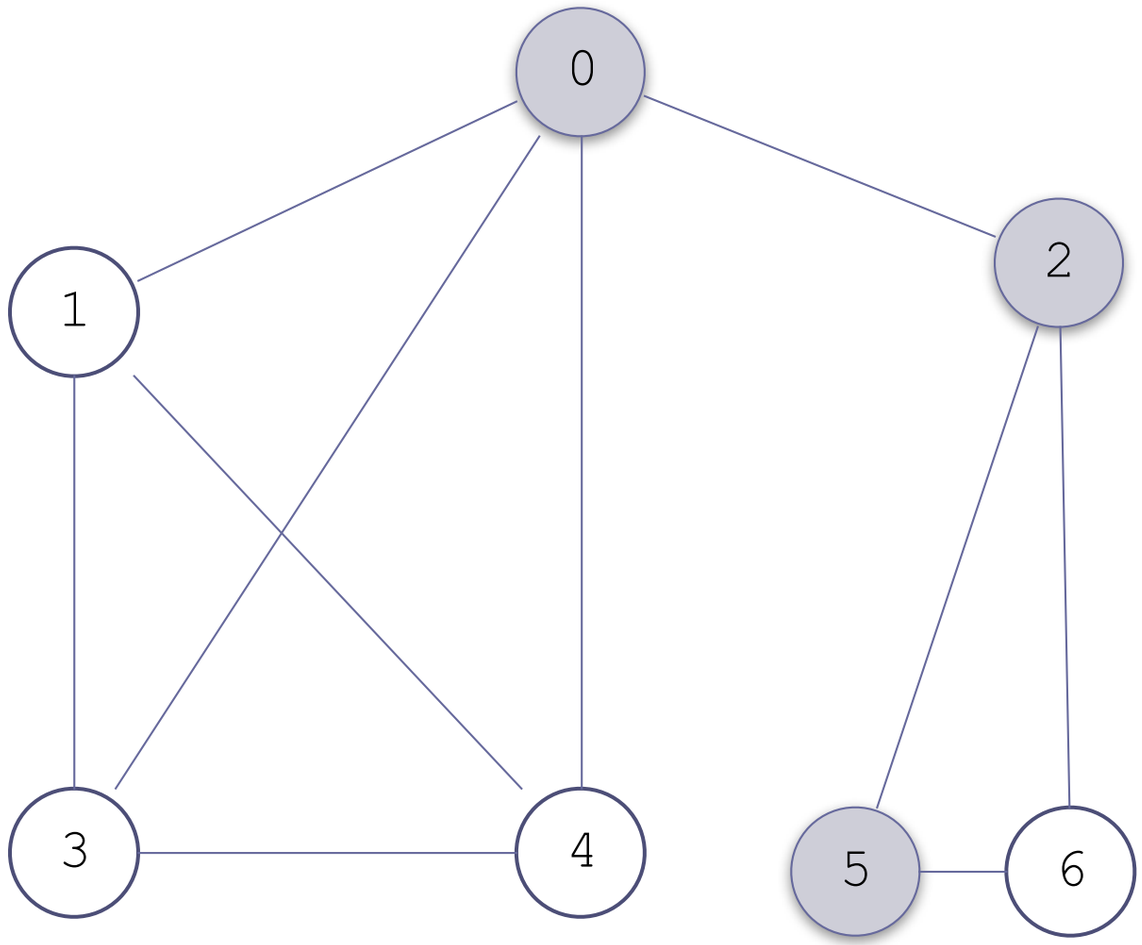


visited

being visited

Example of a Depth-First Search (cont.)

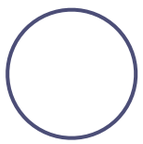
There are no vertices adjacent to 6 not being visited; mark 6 as visited



Discovery (Visit) order:
0, 1, 3, 4, 2, 5, 6

Finish order:
4, 3, 1, 6

unvisited

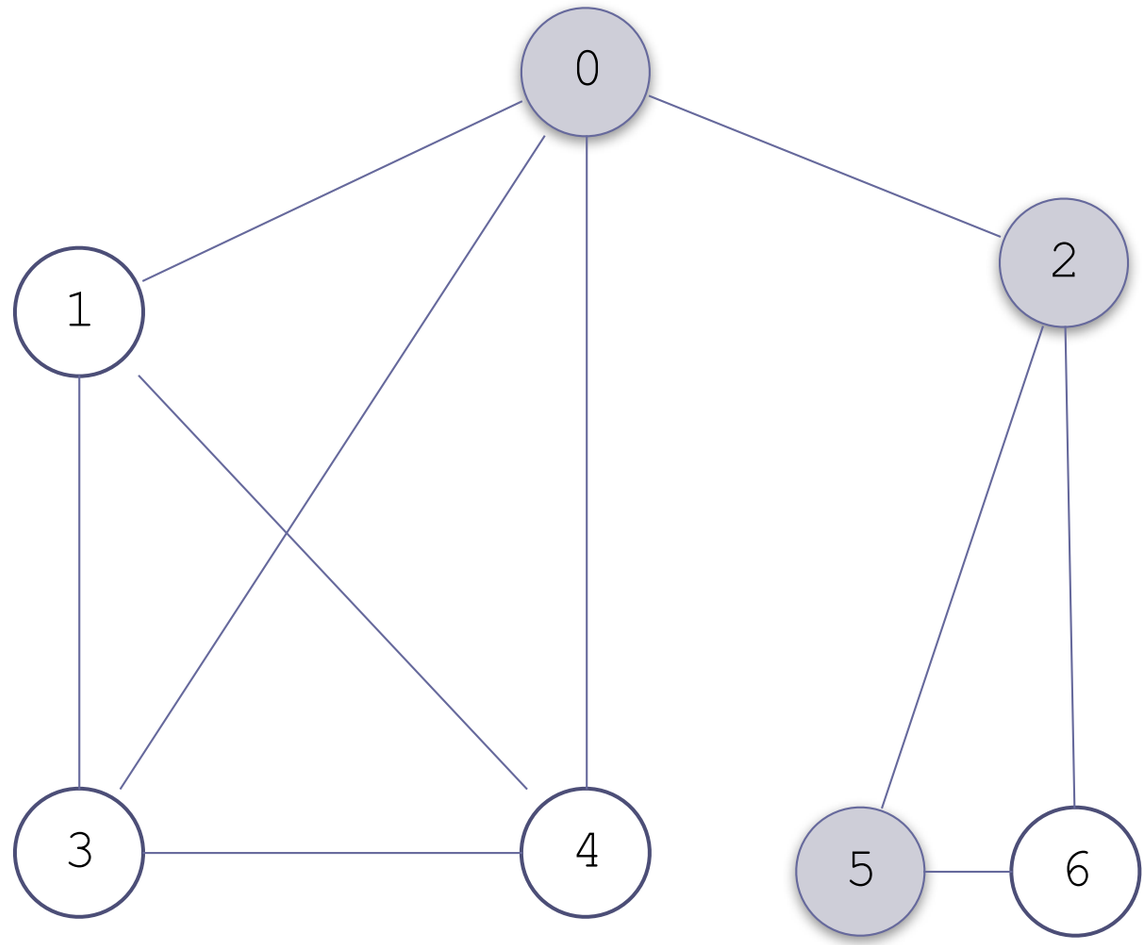


visited

being visited

Example of a Depth-First Search (cont.)

Return from the recursion to 5



Finish order:
4, 3, 1, 6

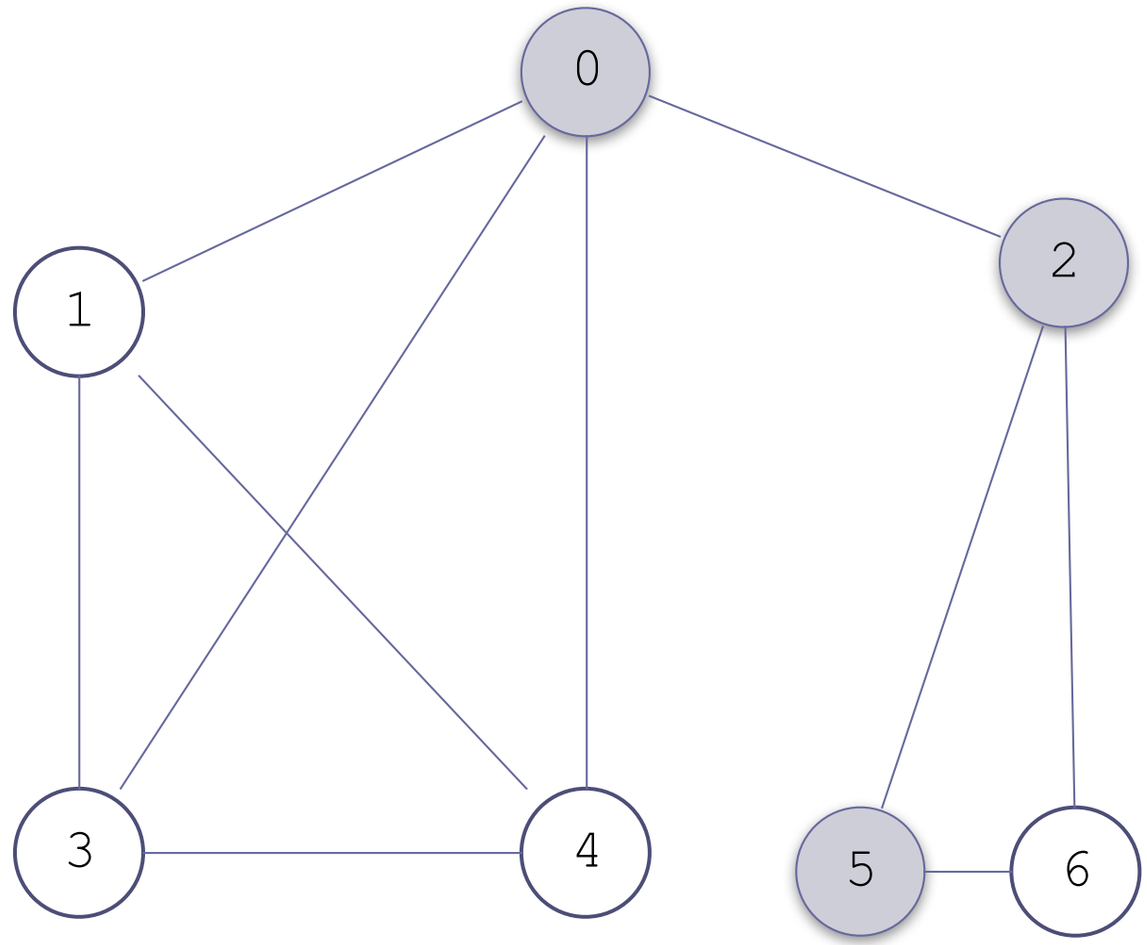
0 **unvisited**

visited

0 **being visited**

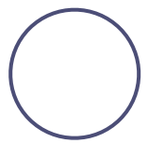
Example of a Depth-First Search (cont.)

Mark 5 as visited



Finish order:
4, 3, 1, 6

unvisited

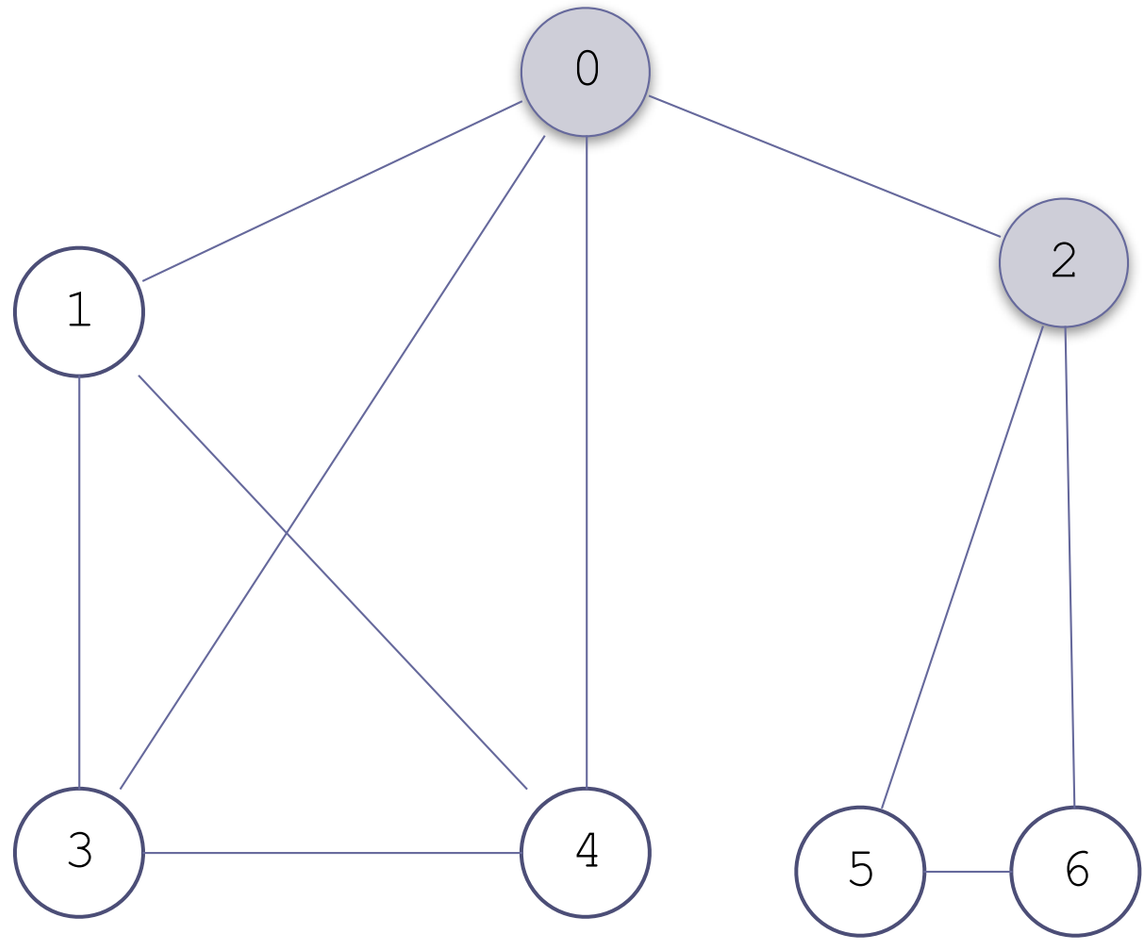


visited

being visited

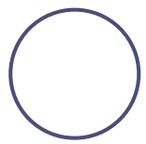
Example of a Depth-First Search (cont.)

Mark 5 as visited



Finish order:
4, 3, 1, 6, 5

unvisited

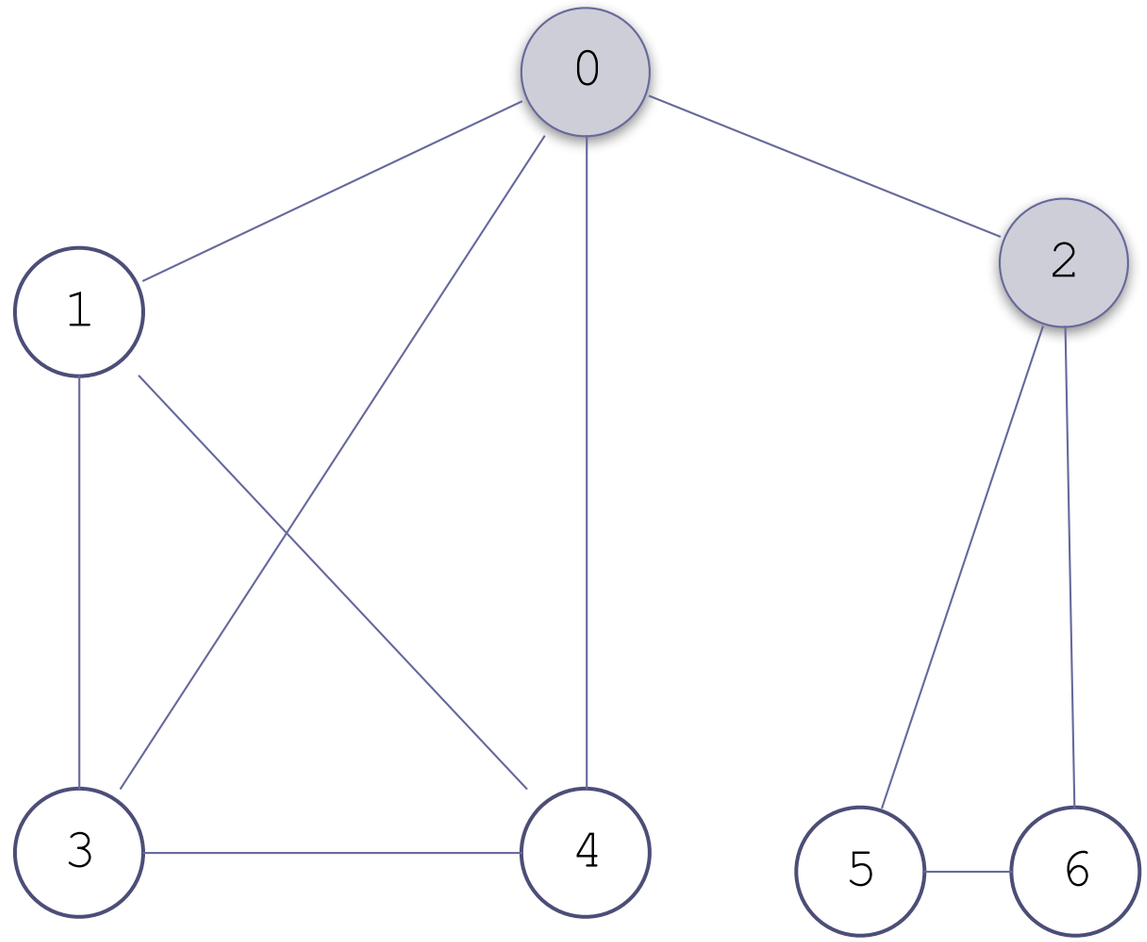


visited

being visited

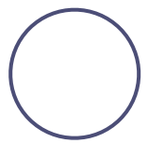
Example of a Depth-First Search (cont.)

Return from the recursion to 2



Finish order:
4, 3, 1, 6, 5

unvisited

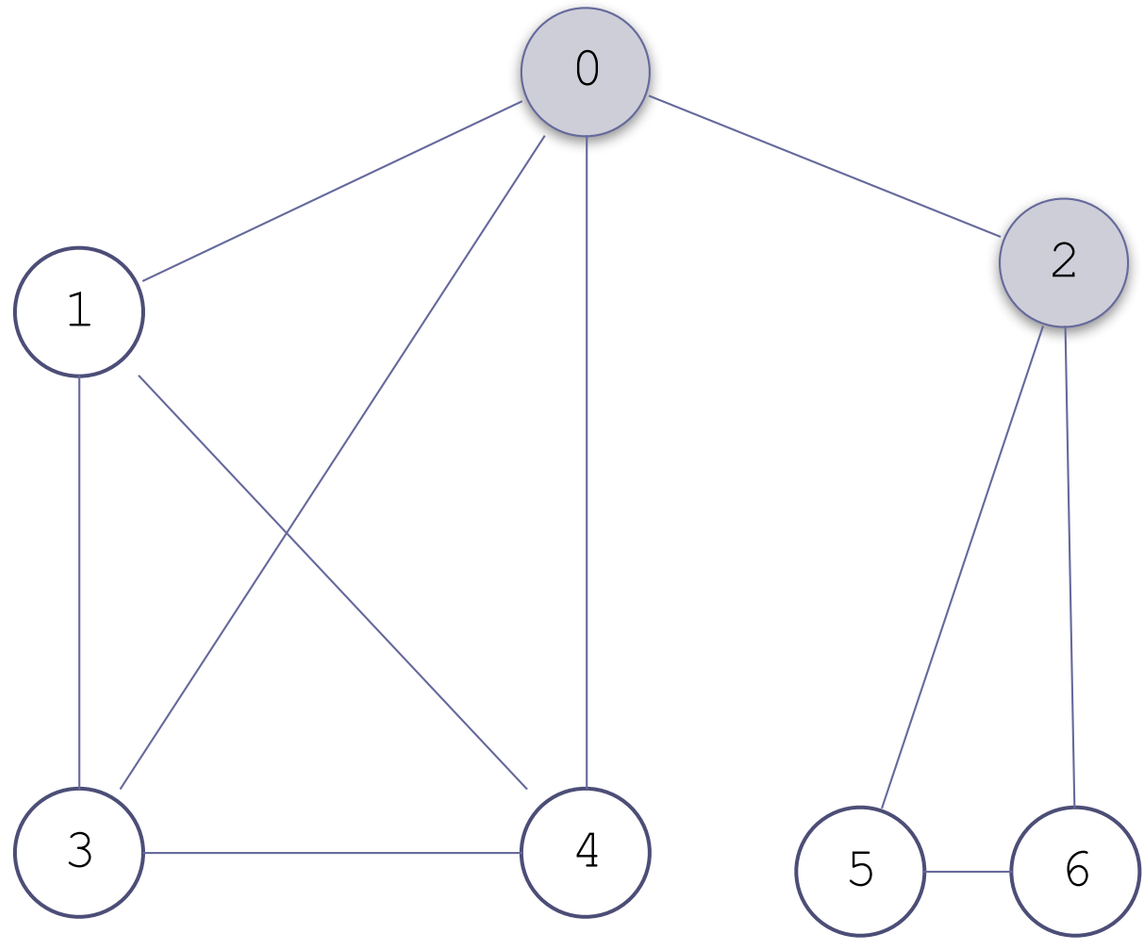


visited

being visited

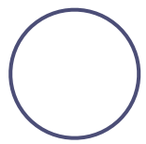
Example of a Depth-First Search (cont.)

Mark 2 as visited



Finish order:
4, 3, 1, 6, 5

unvisited

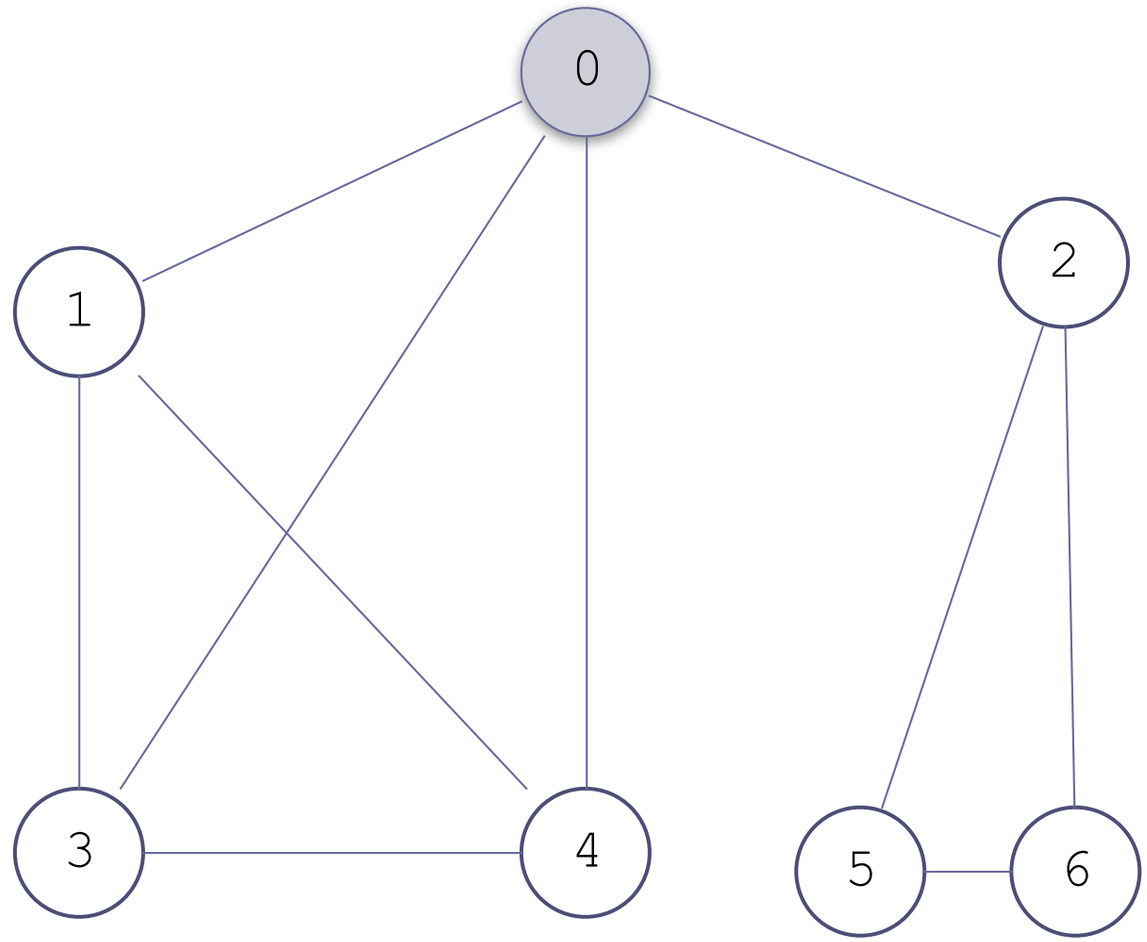


visited

being visited

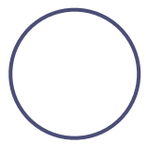
Example of a Depth-First Search (cont.)

Mark 2 as visited



Finish order:
4, 3, 1, 6, 5, 2

unvisited

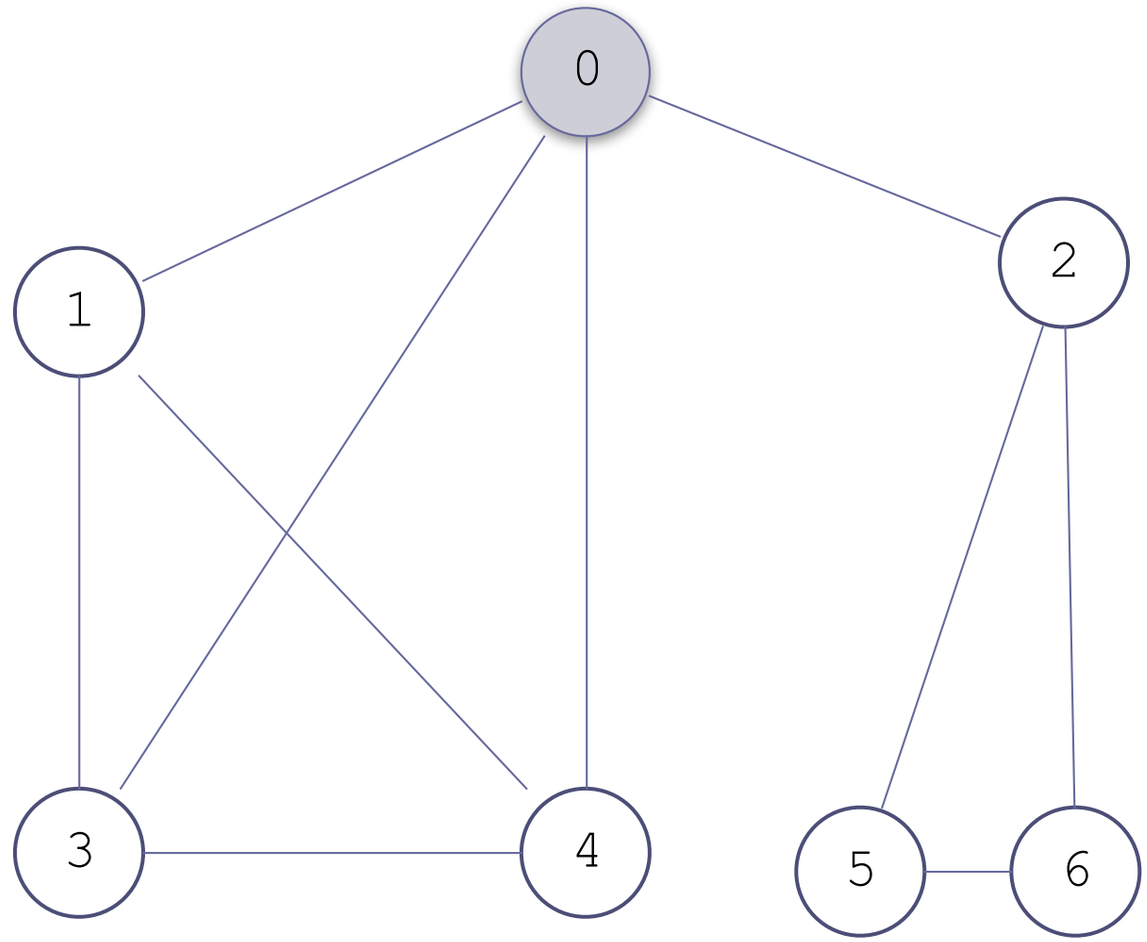


visited

being visited

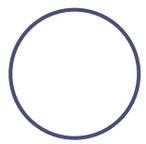
Example of a Depth-First Search (cont.)

Return from the recursion to 0



Finish order:
4, 3, 1, 6, 5, 2

unvisited

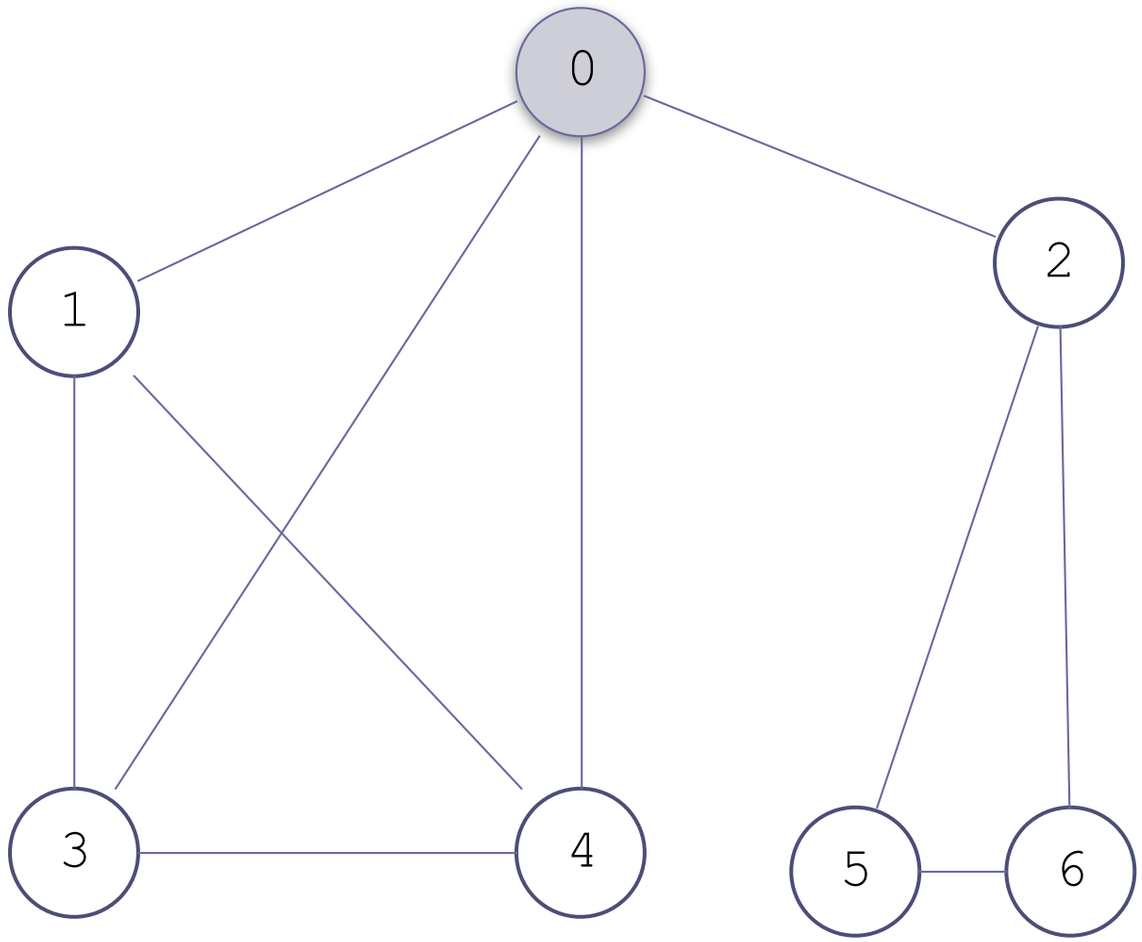


visited

being visited

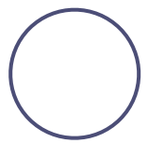
Example of a Depth-First Search (cont.)

There are no nodes adjacent to 0 not being visited



Finish order:
4, 3, 1, 6, 5, 2

unvisited

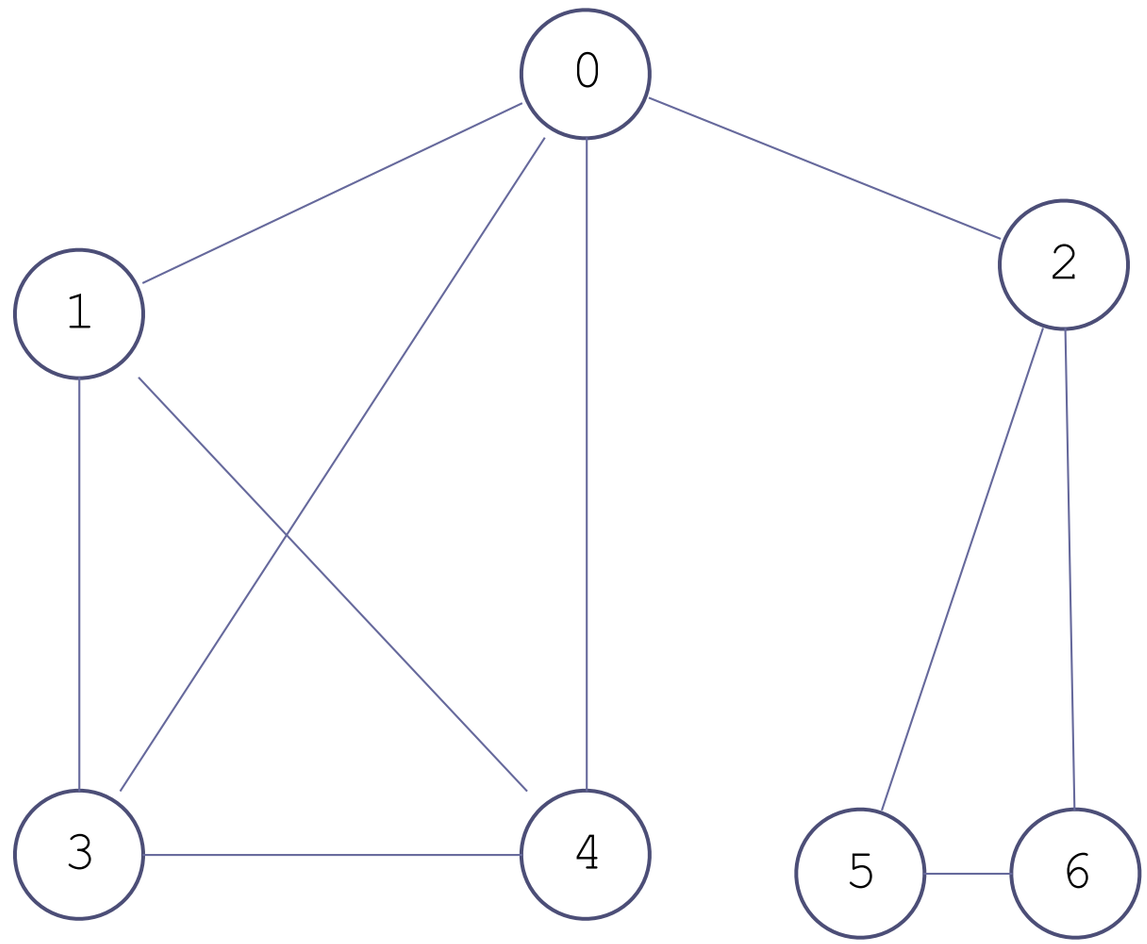


visited

being visited

Example of a Depth-First Search (cont.)

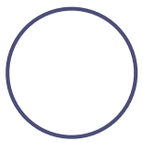
Mark 0 as visited



Discovery (Visit) order:
0, 1, 3, 4, 2, 5, 6, 0

Finish order:
4, 3, 1, 6, 5, 2, 0

unvisited



visited

being visited

Upptäcktsordning

Upptäcktsordningen (discovery order) är den ordning vilken noderna upptäcks:

- 0, 1, 3, 4, 2, 5, 6 i detta exempel

Avslutningsordningen (finish order) är ordningen i vilken noderna avslutas

- 4, 3, 1, 6, 5, 2, 0 i detta exempel
- vi kan lagra bakåtpekare, precis som vi gjorde för bredden-först

Algoritm för bredden-först

Algorithm for Breadth-First Search

1. Take an arbitrary start vertex, mark it identified (color it light blue), and place it in a queue.
2. `while` the queue is not empty
3. Take a vertex, u, out of the queue and visit u.
4. `for` all vertices, v, adjacent to this vertex, u
5. `if` v has not been identified or visited
6. Mark it identified (color it light blue).
7. Insert vertex v into the queue.
8. We are now finished visiting u (color it dark blue).

Algoritm för bredden-först

Algorithm for Breadth-First Search

1. Take an arbitrary start vertex, mark it identified (color it light blue), and place it in a queue.
2. `while` the queue is not empty
3. Take a vertex, u, out of the queue and visit u.
4. `for` all vertices, v, adjacent to this vertex, u
5. `if` v has not been identified or visited
6. Mark it identified (color it light blue).
7. Insert vertex v into the queue.
8. We are now finished visiting u (color it dark blue).

Djupet-först kan implementeras på exakt samma sätt som bredden-först, fast med en stack istället för en kö

Algoritm för djupet -först

Algorithm for Depth -First Search

1. Take an arbitrary start vertex, mark it identified (color it light blue), and place it in a stack
2. `while` the stack is not empty
3. Take a vertex, u, out of the stack and visit u.
4. `for` all vertices, v, adjacent to this vertex, u
5. `if` v has not been identified or visited
6. Mark it identified (color it light blue).
7. Insert vertex v into the stack
8. We are now finished visiting u (color it dark blue).

Djupet-först kan implementeras på exakt samma sätt som bredden-först, fast med en stack istället för en kö

Djupet-först, rekursivt

Algorithm for Depth-First Search

1. Mark the current vertex, *u*, visited (color it light blue), and enter it in the discovery order list
2. **for** each vertex, *v*, adjacent to the current vertex, *u*
3. **if** *v* has not been visited
4. Set parent of *v* to *u*.
5. Recursively apply this algorithm starting at *v*.
6. Mark *u* finished (color it dark blue) and enter *u* into the finish order list.

Men det går lika bra med en rekursiv implementation,
eftersom det är ett sätt att ”dölja” att man använder en stack…

Dessutom kan vi spara avslutningsordningen enkelt
(finish order)

Komplexitet för BFS/DFS

Varje båge testas maximalt en gång

- (två gånger för oriktade grafer)

I värsta fallet blir det alltså |E| tester

- (2 · |E| för oriktade grafer)

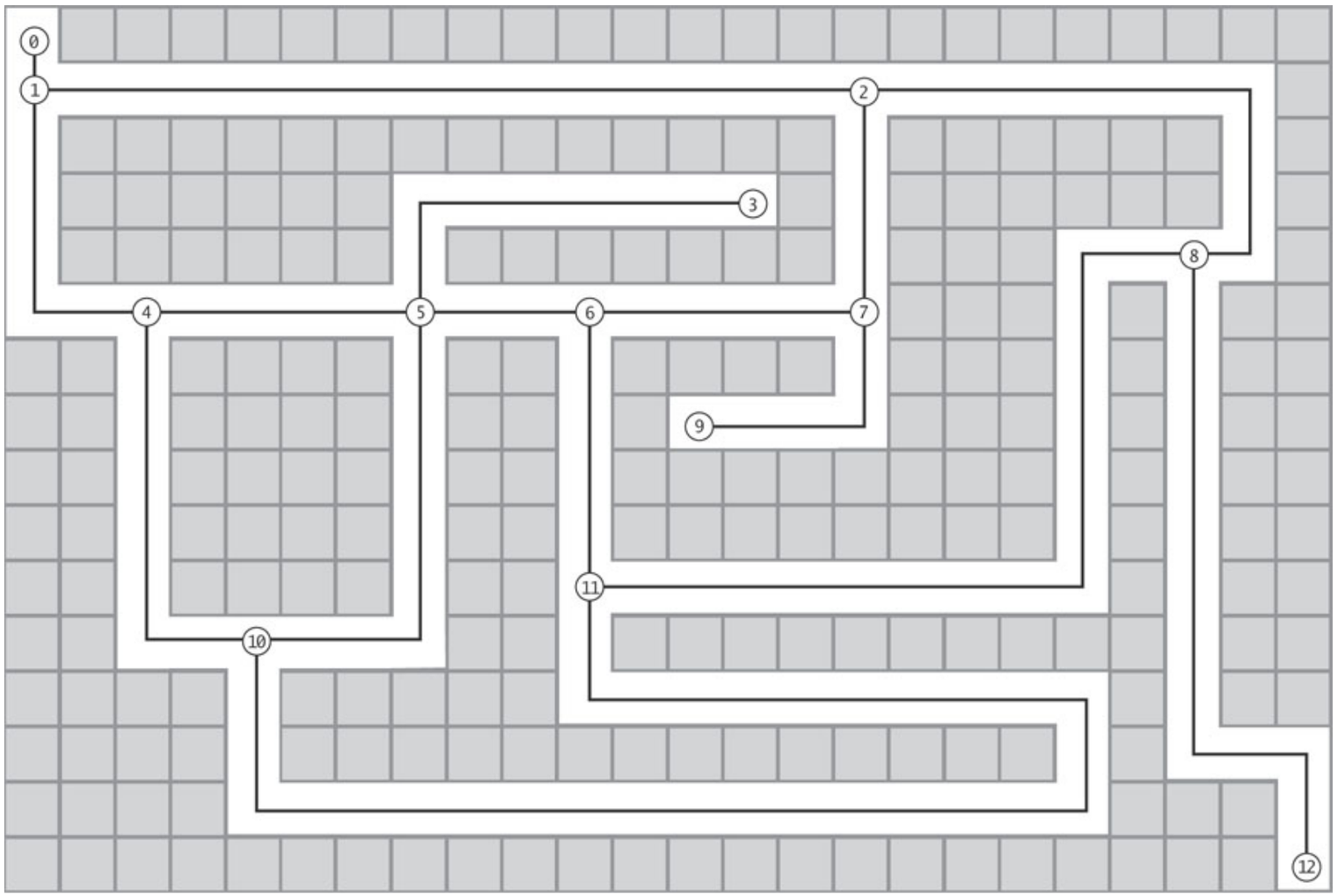
Dvs, komplexiteten är O(|E|)

- (även för oriktade grafer)

Detta gäller både bredden-först och djupet-först

Exempel: Kortaste vägen

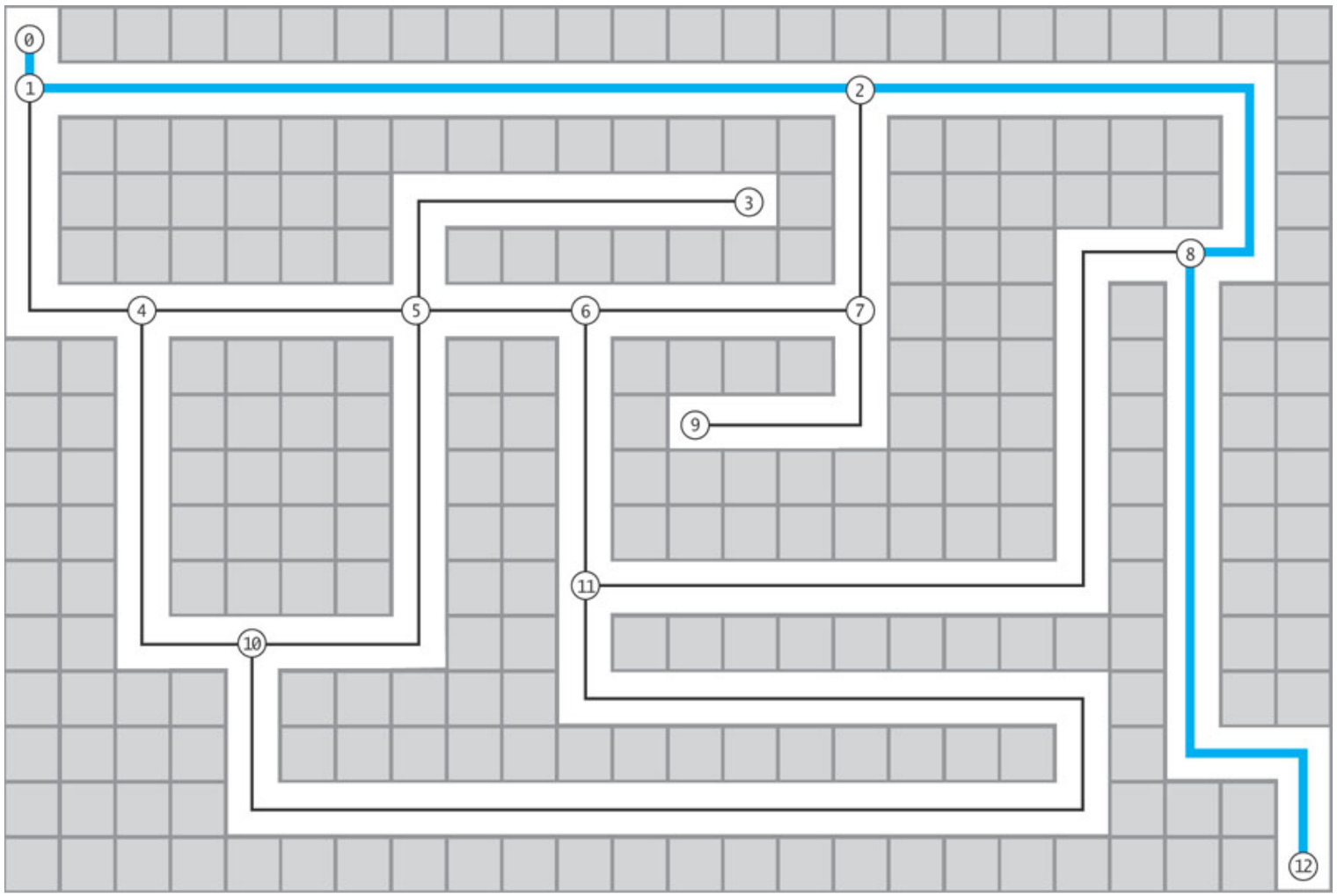
För att hitta kortaste vägen genom en labyrint, kan vi representera labyrinten som en graf:



Exempel: Kortaste vägen

Bredden-först-sökning ger den kortaste vägen:

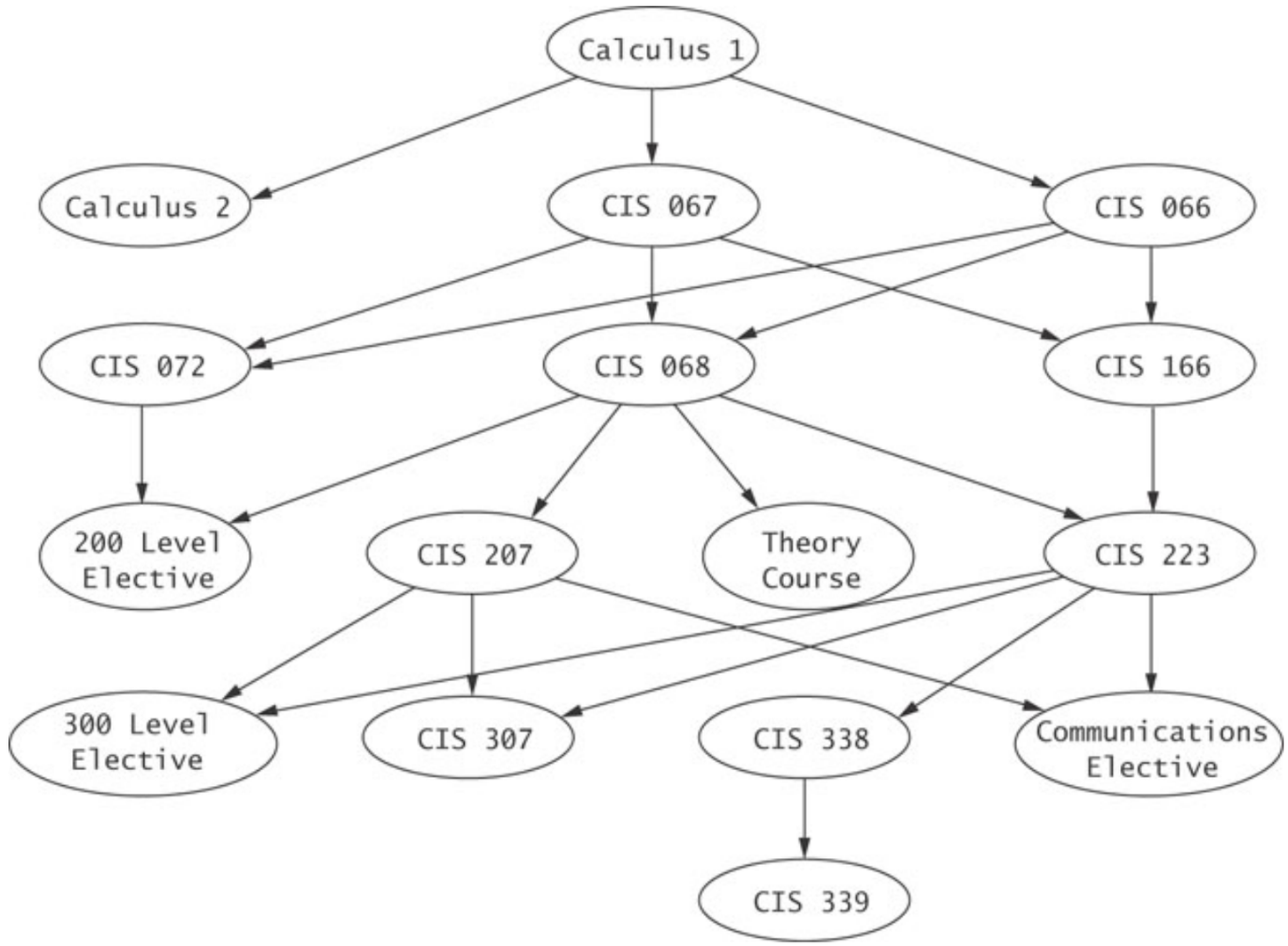
- dvs, minsta antalet korsningar, inte celler!



DAG: Riktade acykliska grafer

Detta är en DAG (directed acyclic graph)

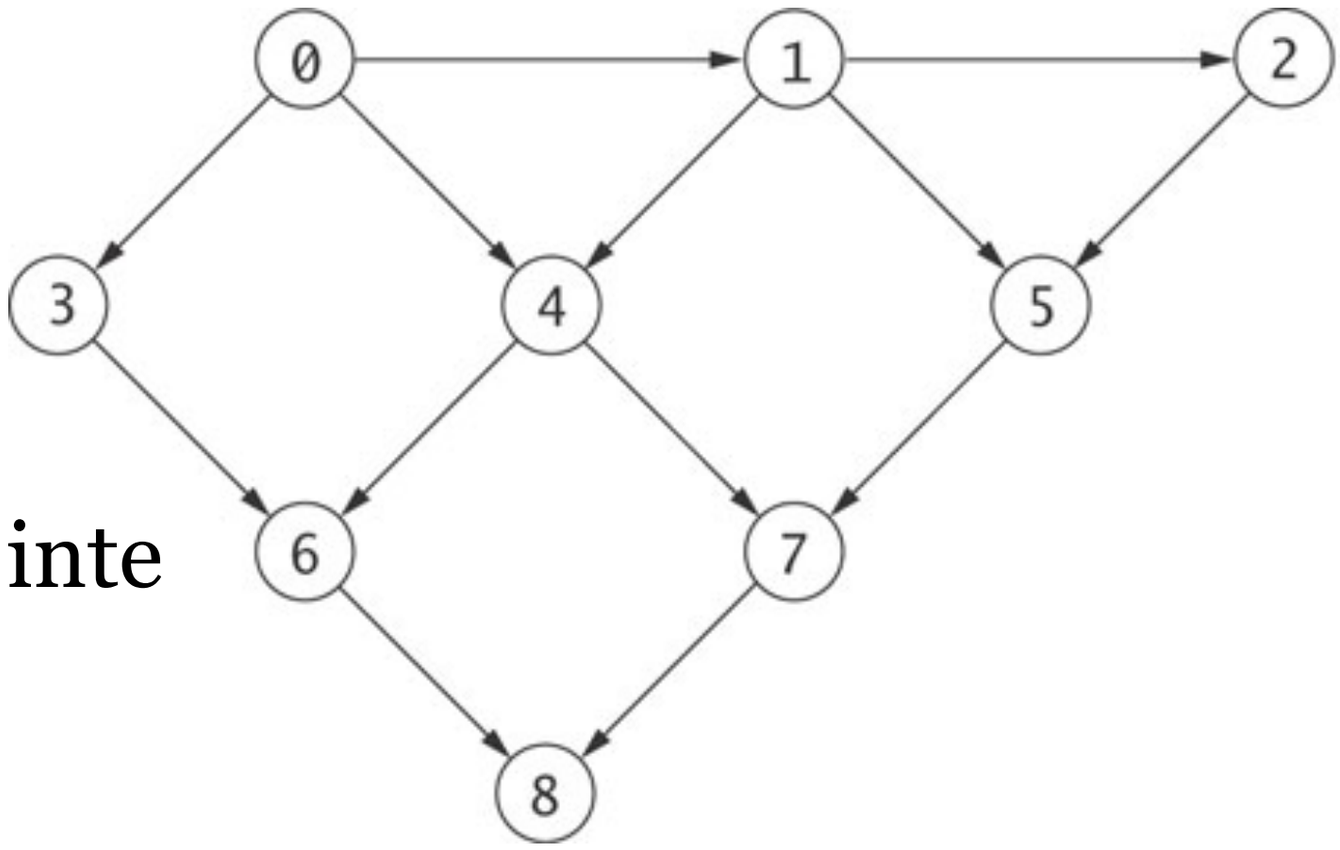
- en riktad graf utan cykler
- i en DAG kan man bara gå framåt, det finns ingen väg tillbaks



Exempel: Topologisk sortering

En topologisk sortering av noderna i en DAG, är att lista noderna i en sådan ordning att

- om (u, v) är en båge, så kommer u före v i listan
- varje DAG har minst en topologisk sortering, ofta fler än en
- 012345678 är en giltig topologisk sortering av denna DAG, men 015342678 är det inte



Exempel: Topologisk sortering

Antag att vi gör en djupet-först-sökning av en DAG:

- om det finns en båge (u, v) i grafen,
- så måste u bli klar efter att v är klar
- dvs, u måste komma efter v i avslutningsordningen

En enkel algoritm för topologisk sortering blir alltså:

- gör en djupet-först-sökning av grafen
- lista noderna i omvänd avslutningsordning