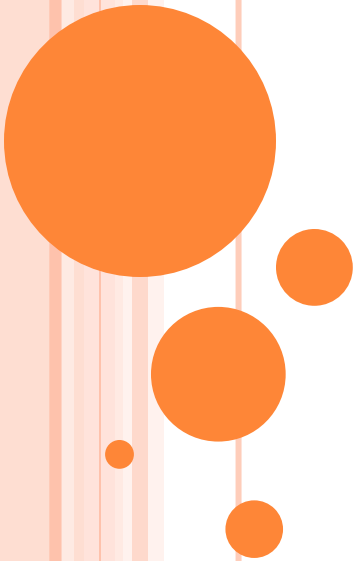
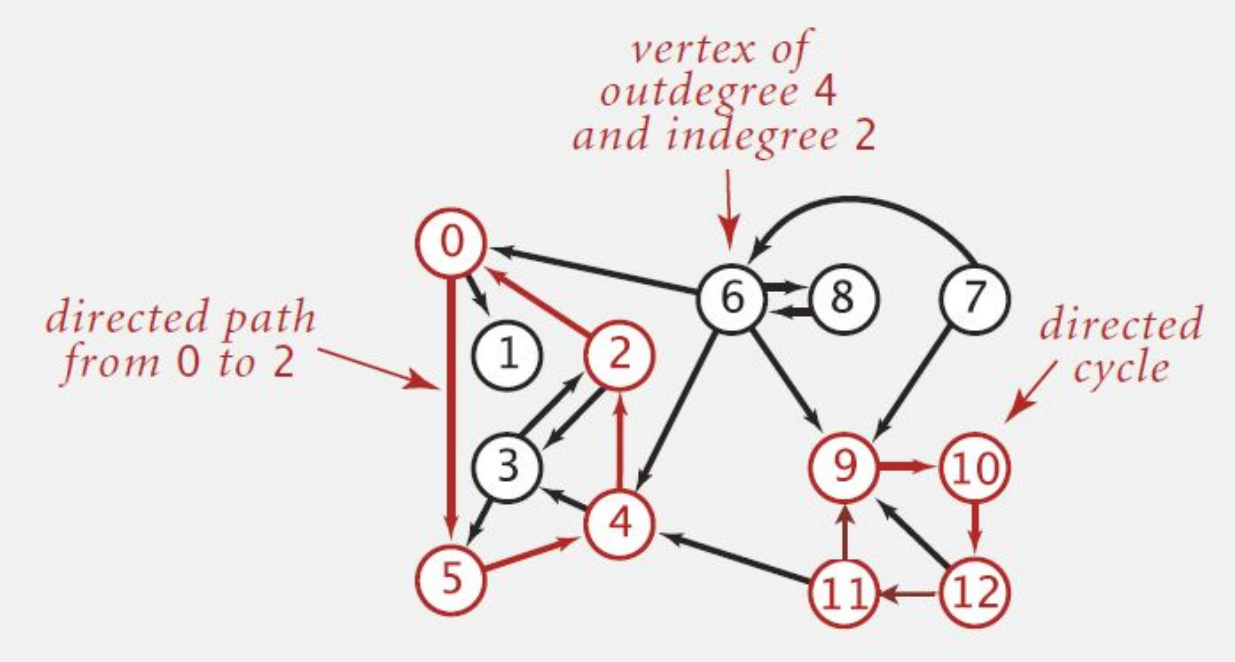


# Лекція 12. Орієнтовані графи

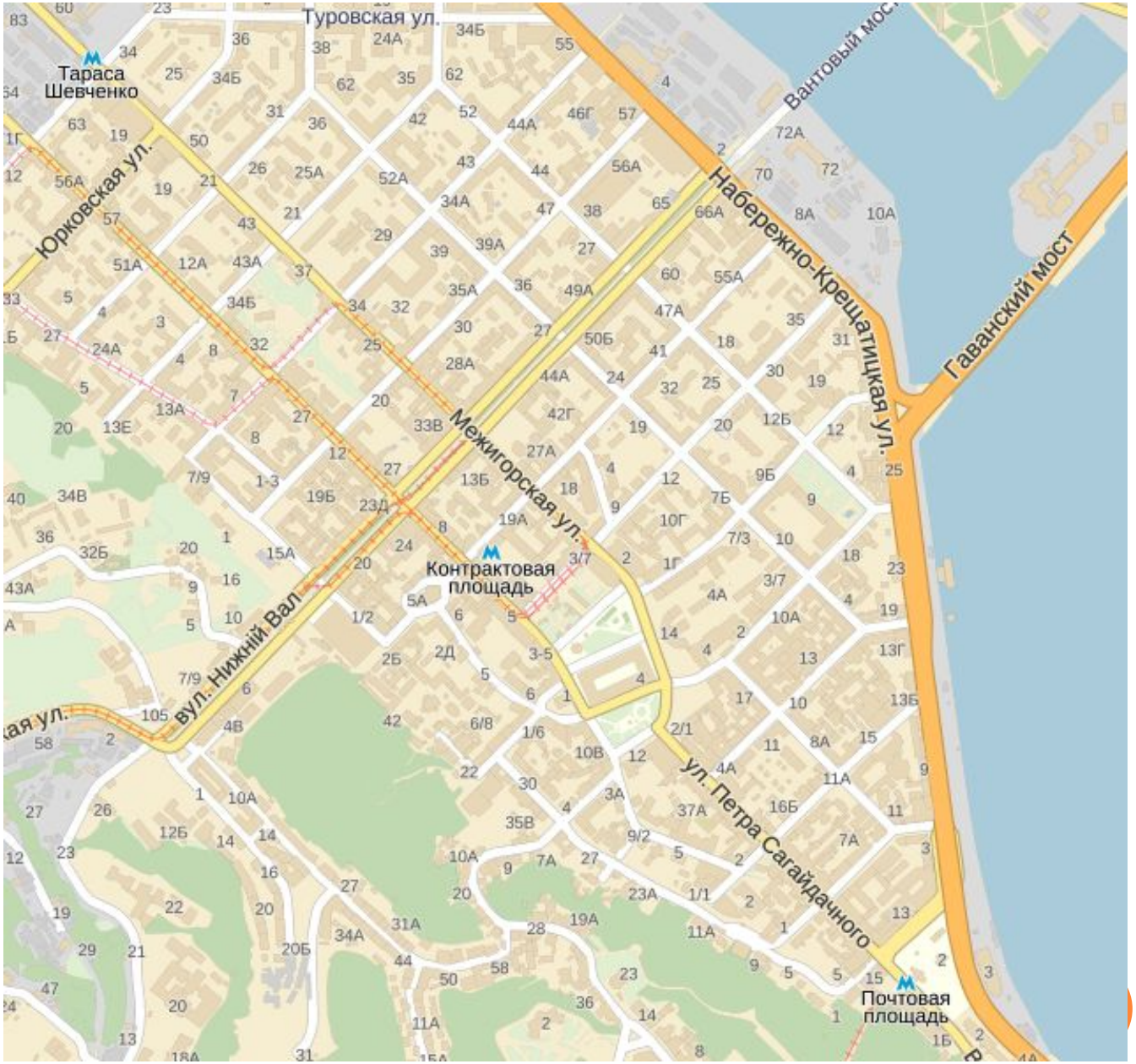
**Глибовець А.М.**

# Вступ

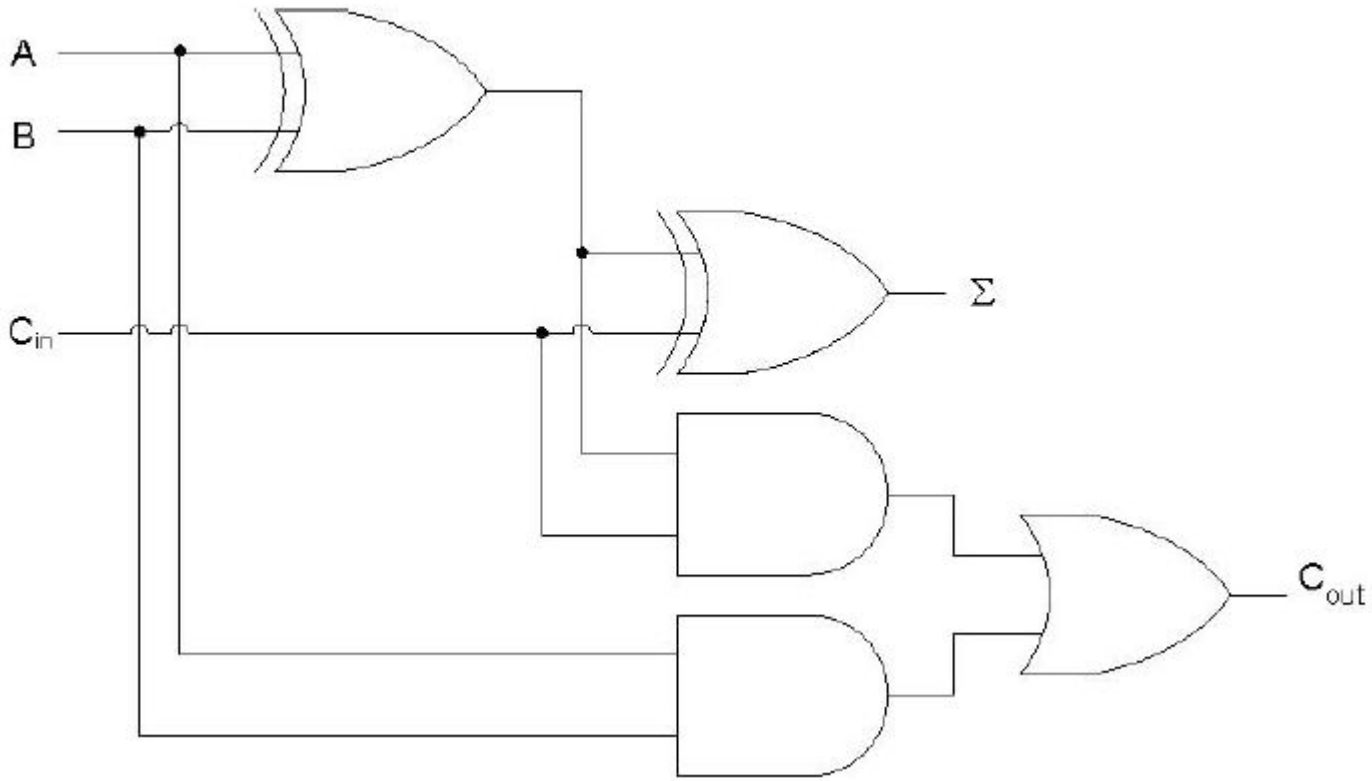
- Орієнтований граф (коротко орграф, англ. digraph) — (мульти) граф, ребрам якого присвоєно напрямок.
- Орієнтовані ребра називаються також дугами.



# Вступ


Туровская ул.
Тараса Шевченко
Юрковская ул.
Межигорская ул.
Набережно-Крещатицкая ул.
Гаванский мост
Вантовый мост
Контрактовая площадь
вул. Нижній Вал
ул. Петра Сагайдачного
Почтовая площадь

# Вступ

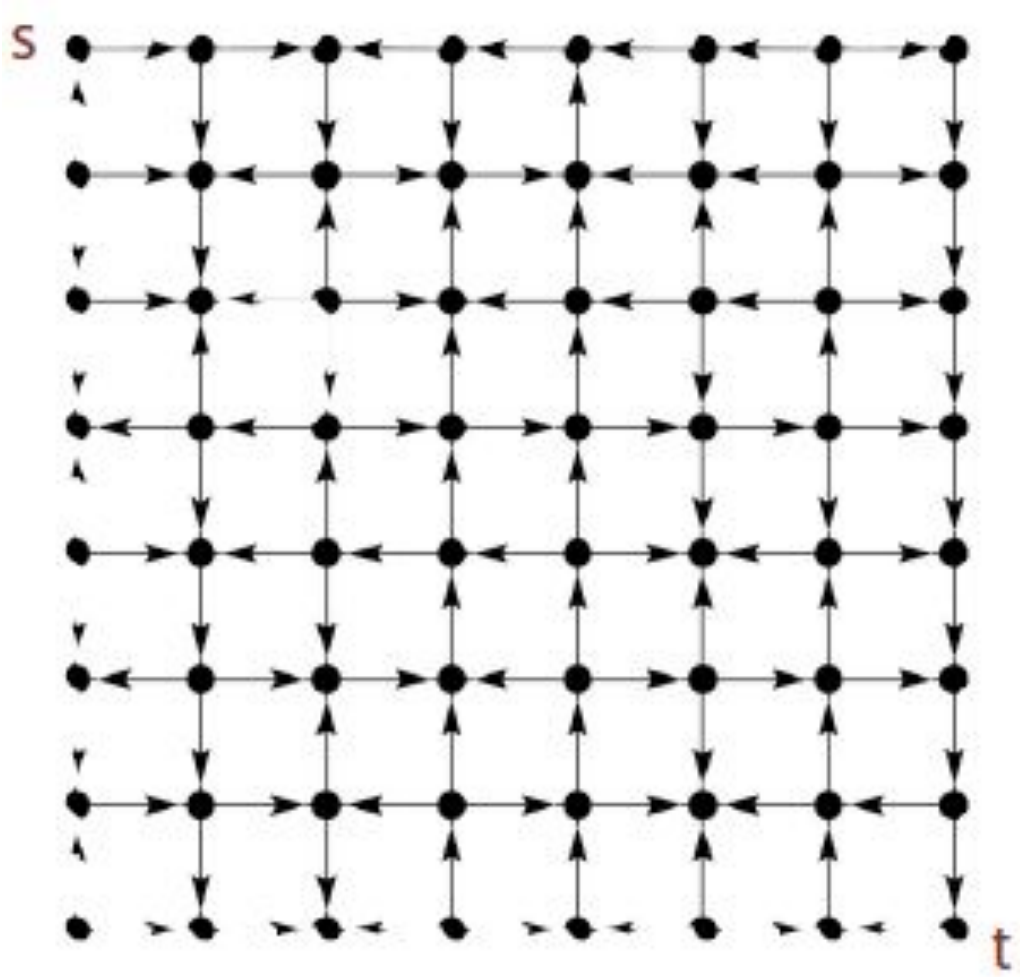


# Вступ

| Орграф | Вершина | Орієнтоване ребро |
|---|---|---|
| Трансопртний | перетин вулиць | вулиці з одностороннім рухом |
| Web | веб-сторінка | гіперлінки |
| Фінансовий | банк | транзакція |
| Телефонний | особа | дзвінок |
| Гра | позиція в грі | доступні кроки |
| Об’єктів | об’єкт | вказівник |

# Задачі

- Пошук шляху. Чи існує шлях з s в t?



# Задачі

- Найкоротший шлях
  - який найкоротший шлях з s в t?
- Топологічне сортування
  - Чи можете ви зобразити орграф так щоб всі ребра були направлені догори
- Сильна зв’язність
  - чи існує шлях між всіма парами вершин орграфа
- Транзитивне замикання
  - для яких вершин v і w, існує шлях від v до w
- PageRank
  - важливість сторінок

# Digraph API

- public class Digraph
  - Digraph(int V) //створити порожній орграф з V вершин
  - Digraph(In in) // створити орграф з вхідного потоку
  - void addEdge(int v, int w) // додати орієнтоване ребро v->w
  - Iterable<Integer> adj(int v) //вершини з'єднані (сусідні) з v
  - int V() //кількість вершин
  - int E() //кількість ребер

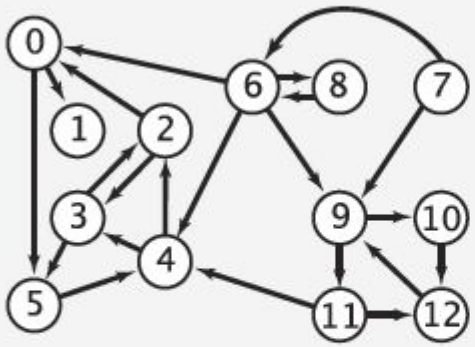
- In in = new In(args[0]);
- Digraph G = new Digraph(in); //читаємо граф з вхідного потоку
- for (int v = 0; v < G.V(); v++) //виводимо кожне ребро (один раз)
  - for (int w : G.adj(v))
    - StdOut.println(v + "->" + w);

# Digraph API

tinyDG.txt

V → 13
E → 22

```
 4  2
 2  3
 3  2
 6  0
 0  1
 2  0
11 12
12  9
 9 10
 9 11
 7  9
10 12
11  4
 4  3
 3  5
 6  8
 8  6
 ⋮
```

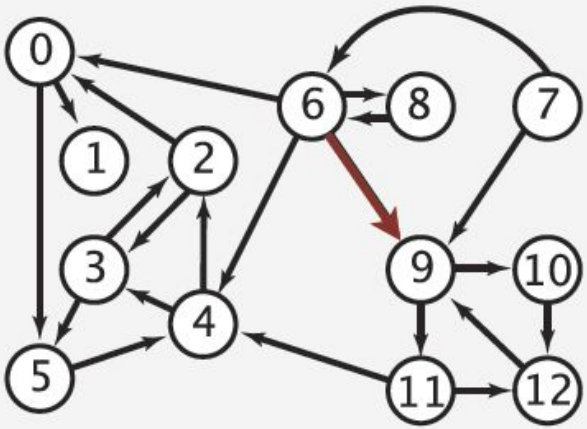


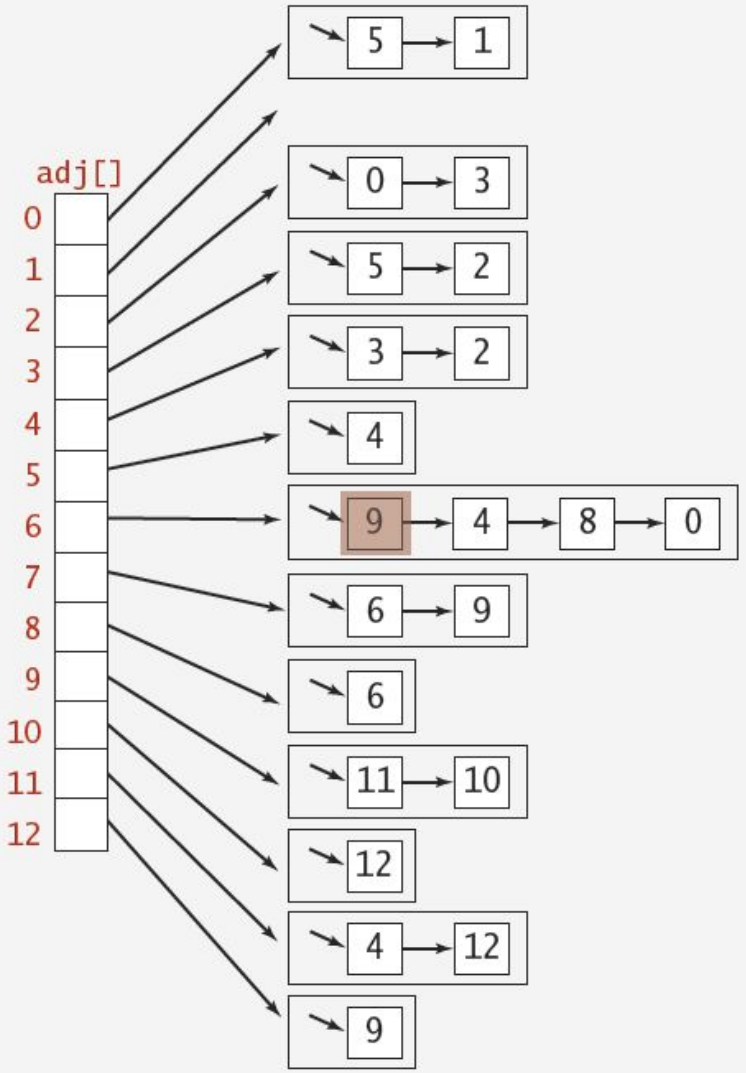

```
% java Digraph tinyDG.txt
0->5
0->1
2->0
2->3
3->5
3->2
4->3
4->2
5->4
⋮
11->4
11->12
12-9
```

# Внутрішнє представлення орграфа

- Як ви думаєте, що ми маємо використати для внутрішнього представлення орієнтованого графа?

# Внутрішнє представлення орграфа



# Реалізація Digraph

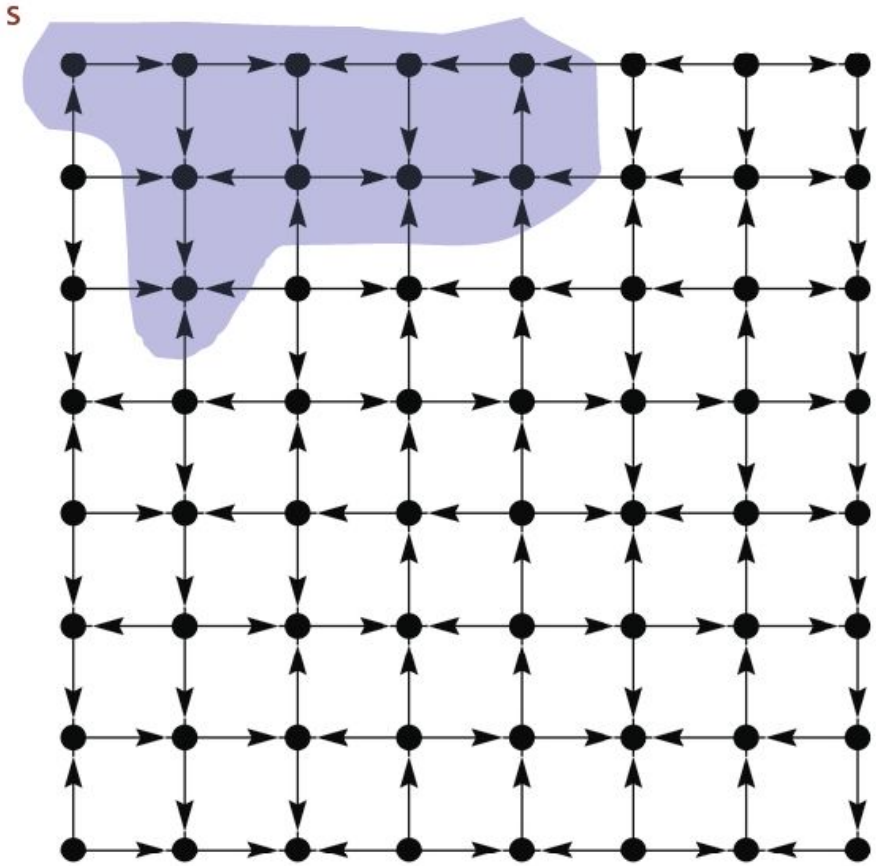
- Що необхідно змінити, що б отримати реалізацію Digraph?
- public class Graph{
  - private final int V;
  - private final Bag<Integer>[] adj;
  - public Graph(int V){
    - this.V = V;
    - adj = (Bag<Integer>[]) new Bag[V];
    - for (int v = 0; v < V; v++)
    - adj[v] = new Bag<Integer>();
  - }
  - public void addEdge(int v, int w){
    - adj[v].add(w);
    - adj[w].add(v);
  - }
  - public Iterable<Integer> adj(int v){
    - return adj[v];
  - }
- }

# Реалізація Digraph

```
public class Digraph{
    private final int V;
    private final Bag<Integer>[] adj;
    public Digraph(int V){
        this.V = V;
        adj = (Bag<Integer>[]) new Bag[V];
        for (int v = 0; v < V; v++)
            adj[v] = new Bag<Integer>();
    }
    public void addEdge(int v, int w){
        adj[v].add(w);
        //прибрали стрічку
    }
    public Iterable<Integer> adj(int v){
        return adj[v];
    }
}
```

# Досяжність

- Проблема. Знайти всі вершини досяжні з s в орієнтованому графі.



# Алгоритми пошуку

- Ми вже знаємо два алгоритми пошуку,
- нагадайте мені.
- Чи можна їх використовувати для орієнтованих графів?

# DFS

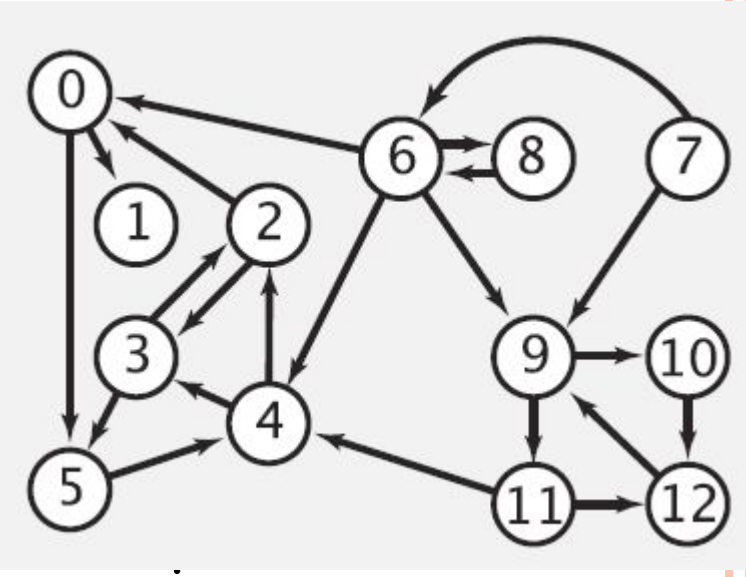
- DFS(v) – відвідати вершину v
  - відмічаємо v як відвідану
  - рекурсивно відвідуємо усі суміжні невідмічені вершини.
- Кожний неорієнтований граф є орграфом (з ребрами направленими в обидва боки).
- Що необхідно змінити в реалізації алгоритму, що б він працював на орієнтованих графах?

# BFS

- Алгоритм:
  - повторюємо поки черга не порожня:
    - дістати вершину v з черги
    - додати в чергу всі невідвідані вершини суміжні з v і помітити їх
- Реалізація аналогічна.

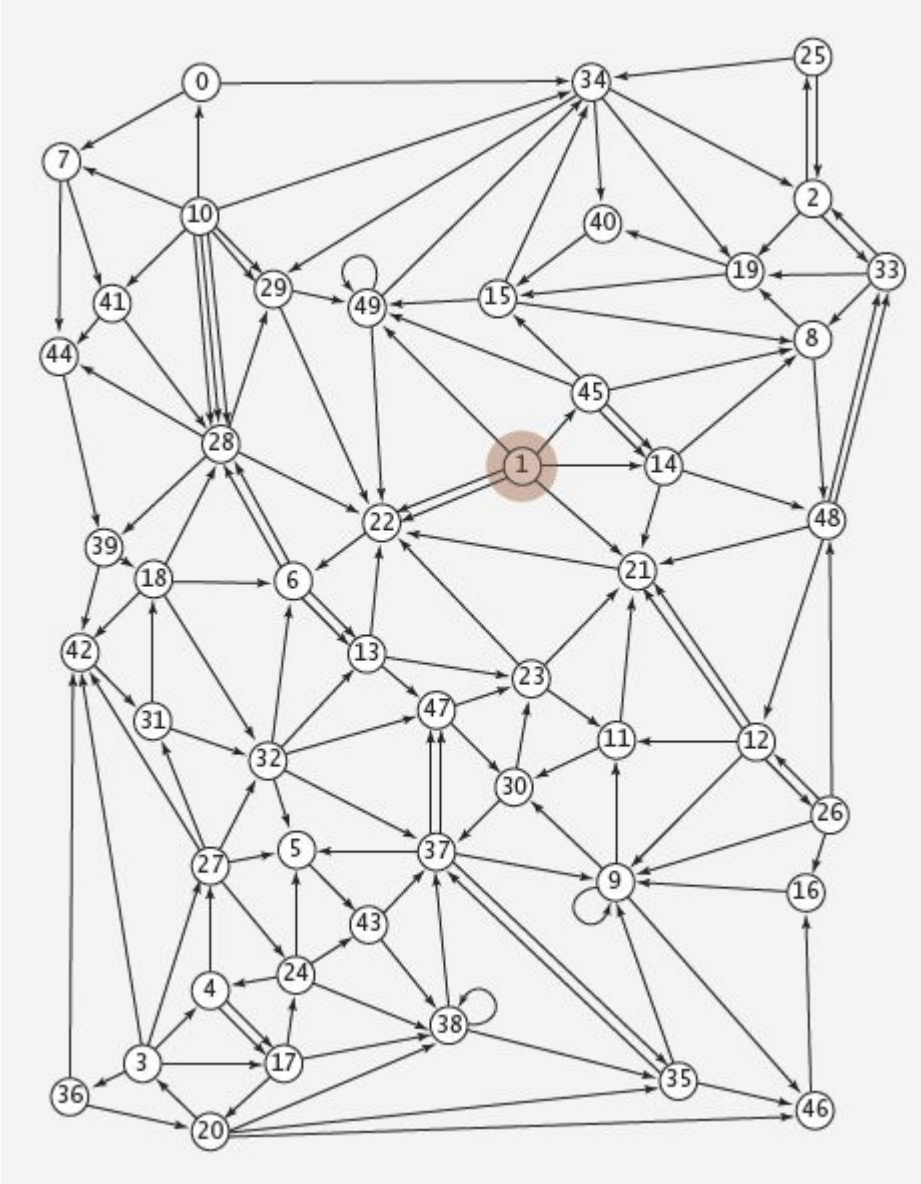
# Найкоротший шлях з множини джерел

- Найкоротший шлях з множини джерел.
  - Маємо орграф і множину джерел вершин, знайти найкоротший шлях до заданої вершини.
- Приклад. S={1,7,10}
  - Найкоротший шлях до 4:
    - 7->6->4
  - Найкоротший шлях до 5:
    - 7->6->0->5
  - Найкоротший шлях до 12:
    - 10->12
- Як реалізувати?
- Беремо BFS, але ініціалізувати чергу всіма вершинами множини.



# Обхід веб

- Пошуковий робот.
  - Розглянемо спрощену задачу.
    - На вхід подається початкова сторінка.
    - Віднайти всі гіперлінки на сторінці і продовжити обхід



# Обхід веб

- Алгоритм. Використовуємо BFS.
  - вибираємо початкову сторінку як s
  - створюємо чергу веб-сайтів для відвідування
  - створюємо чергу вже відвіданих сторінок
  - з черги забираємо наступний сайт для відвідування і додаємо в чергу сайти на які є посилання з сторінки.
- Чому не використати DFS?

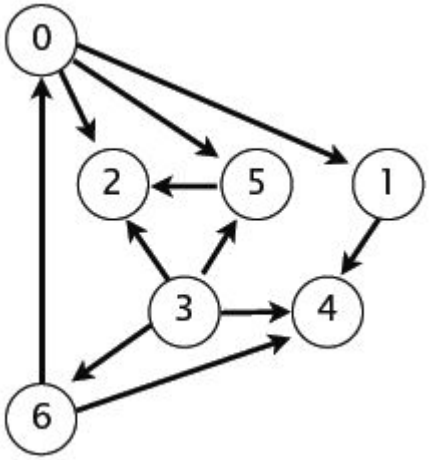
# Обхід веб

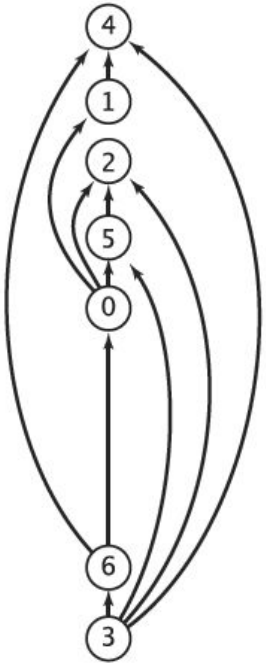
- Реалізація WebCrawler.java

# Планування

- Нам даються певні завдання, що мають бути виконані з черговістю.
- Питання, в якій черзі ми маємо робити ці завдання.
- Для цього використаємо модель орграфа.
  - вершини – завдання
  - ребра - черговість

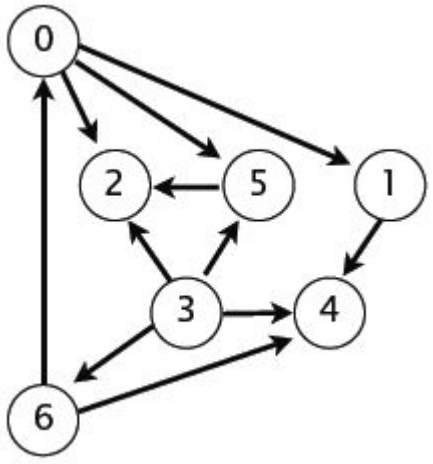
0. Algorithms
1. Complexity Theory
2. Artificial Intelligence
3. Intro to CS
4. Cryptography
5. Scientific Computing
6. Advanced Programming



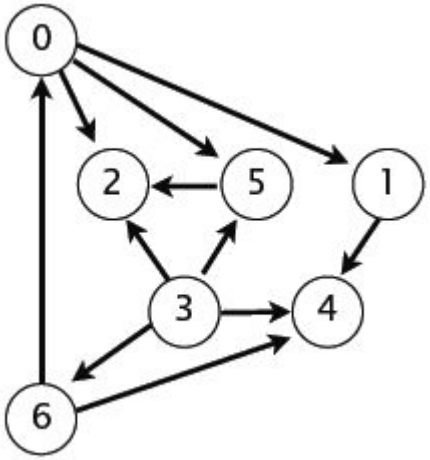
# Топологічне сортування

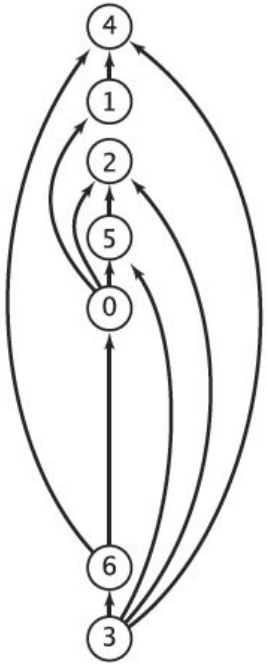
- Топологічний пошук працює на орієнтованих ациклічних графах (ОАГ).
- Якщо в нас буде присутній цикл, ми не зможемо вирішити проблему.
- Чому?



# Топологічне сортування

- Якщо ми маємо ОАГ, для вирішення задачі топологічного сортування ми маємо перемалювати його так, що б всі направлені ребра вказували вгору.
- Таким чином ми отримаємо послідовність дій (порядок виконання).



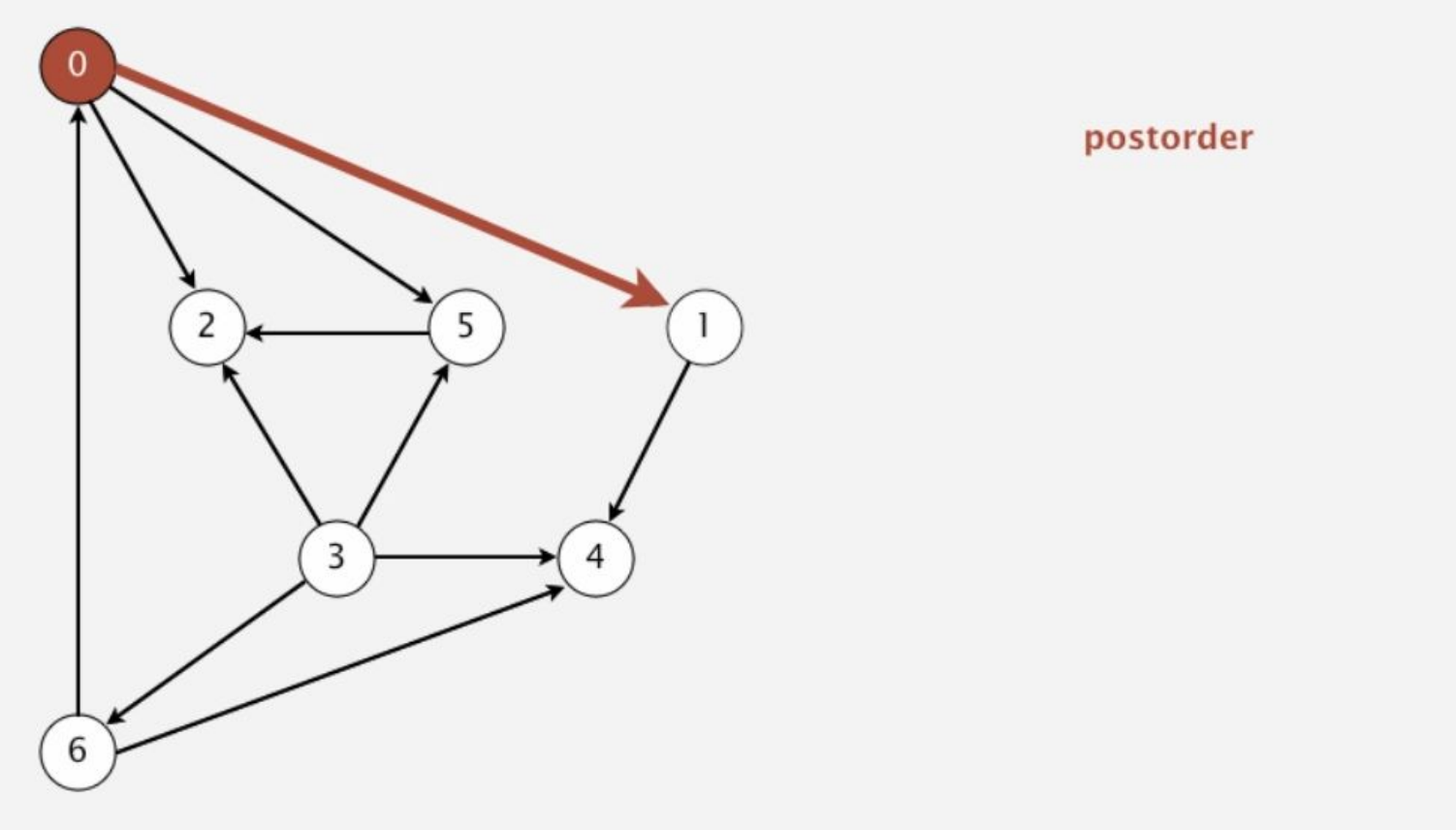



# Топологічне сортування

- Для вирішення цієї задачі ми можемо використати DFS
- Алгоритм:
  - запустити DFS
  - повернути вершини в зворотному порядку

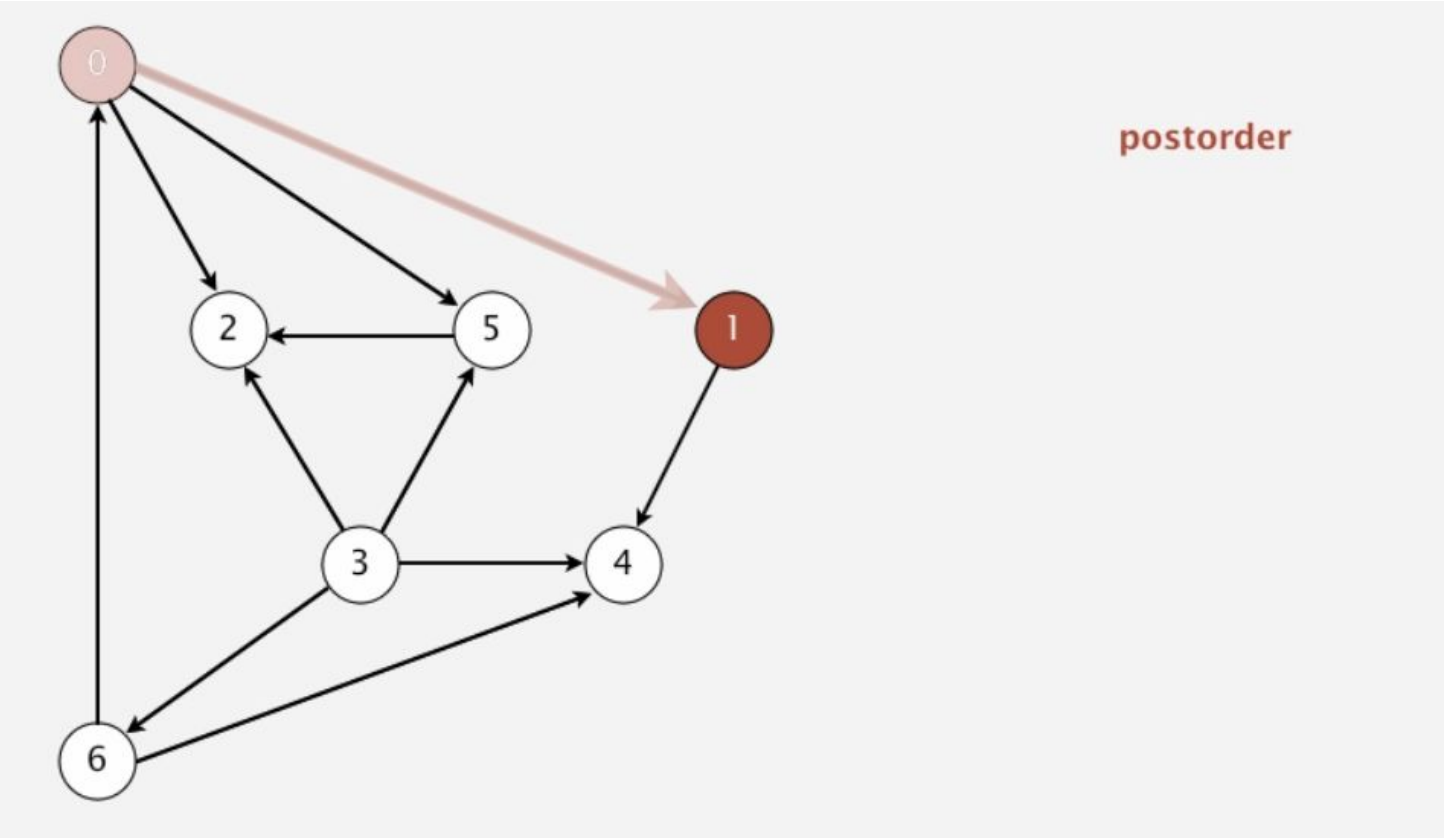
# Топологічне сортування

- Відвідати 0. Відмітити як відвідану вершину
- З 0 є вихідні ребра.



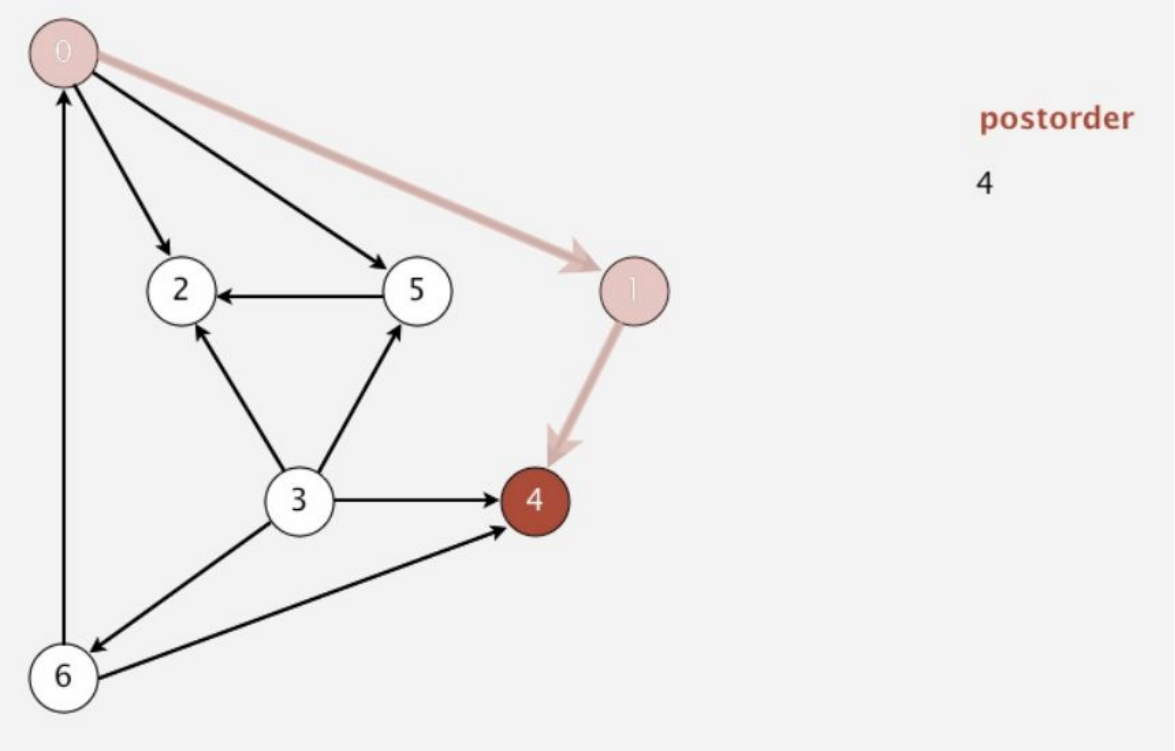
# Топологічне сортування

- Відвідати 1. Відмітити як відвідану.
- З 1 є вихідні ребра.



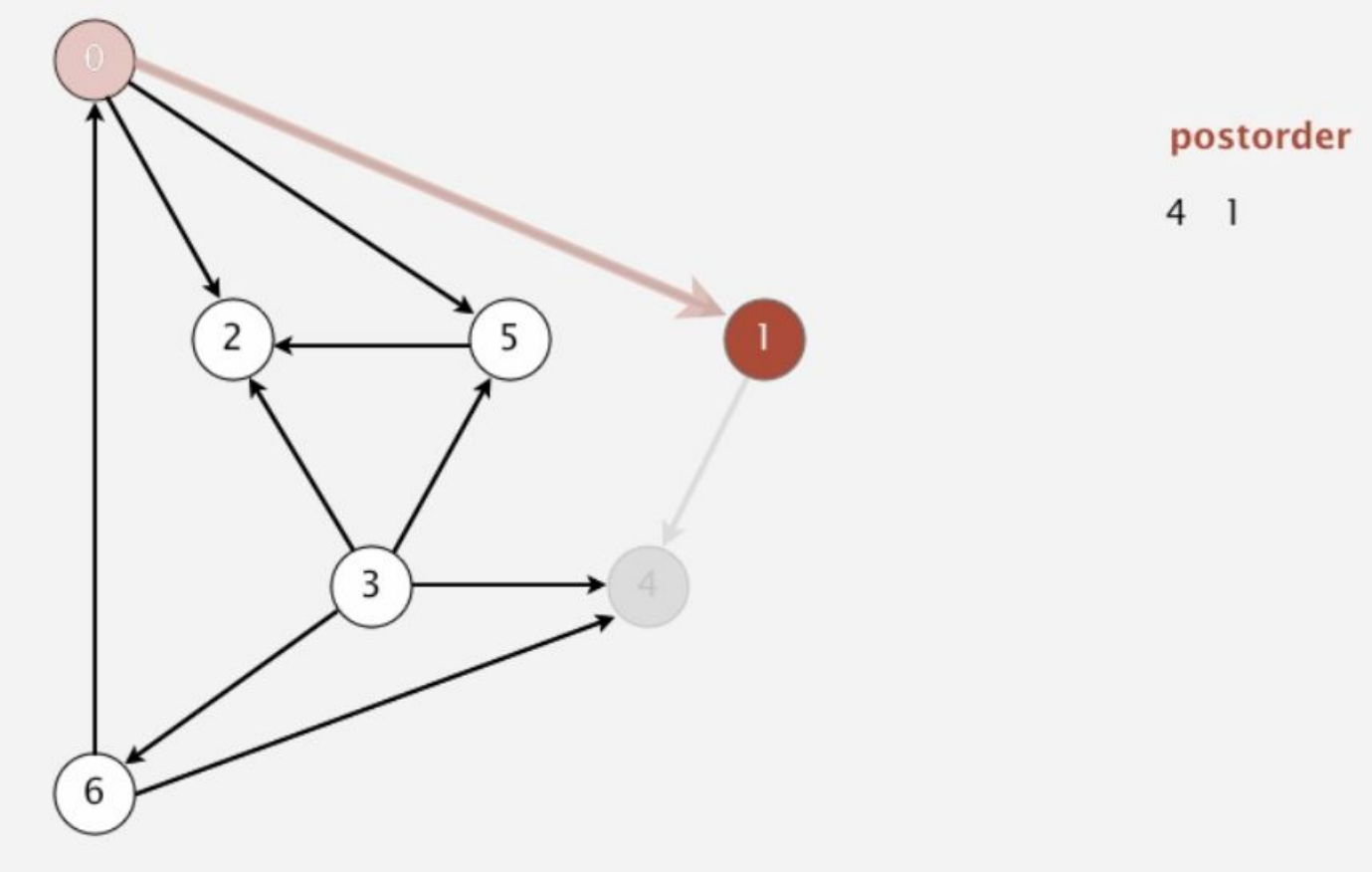
# Топологічне сортування

- Відвідати 4. Відмітити як відвідану.
- З 4 немає вихідних ребер.
- Тому повертаємо 4 і записуємо в postorder.



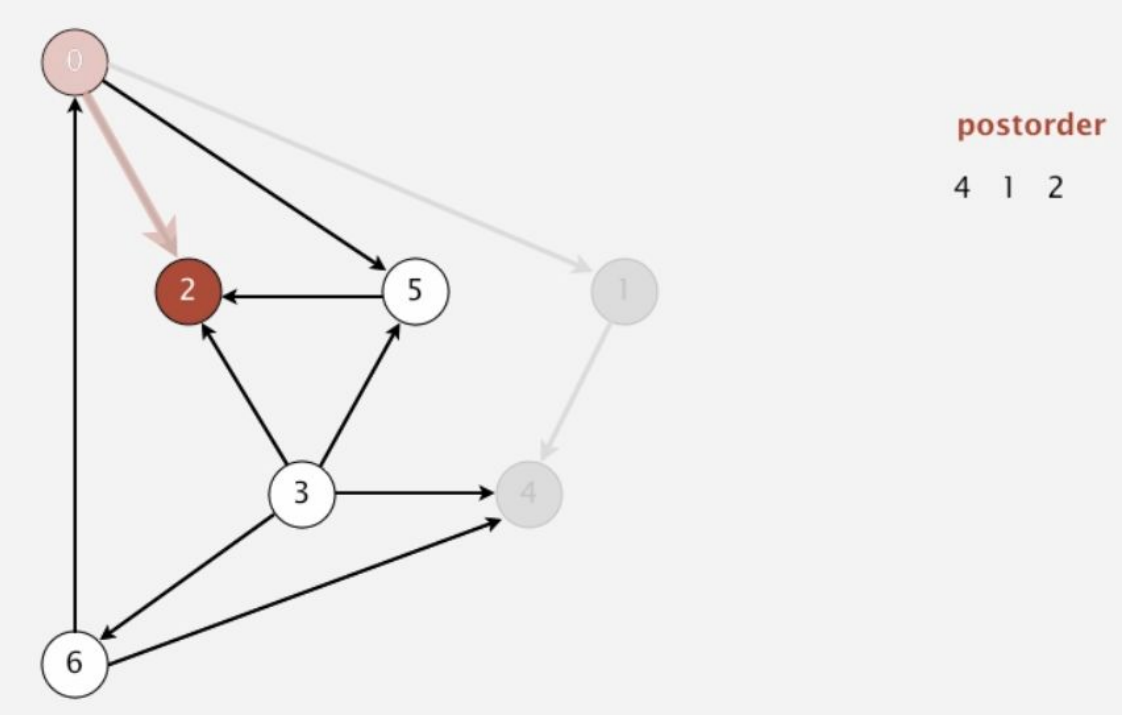
# Топологічне сортування

- Повернулися назад.
- З 1 більше немає вихідних ребер.
- Тому повертаємо 1 і записуємо в postorder.



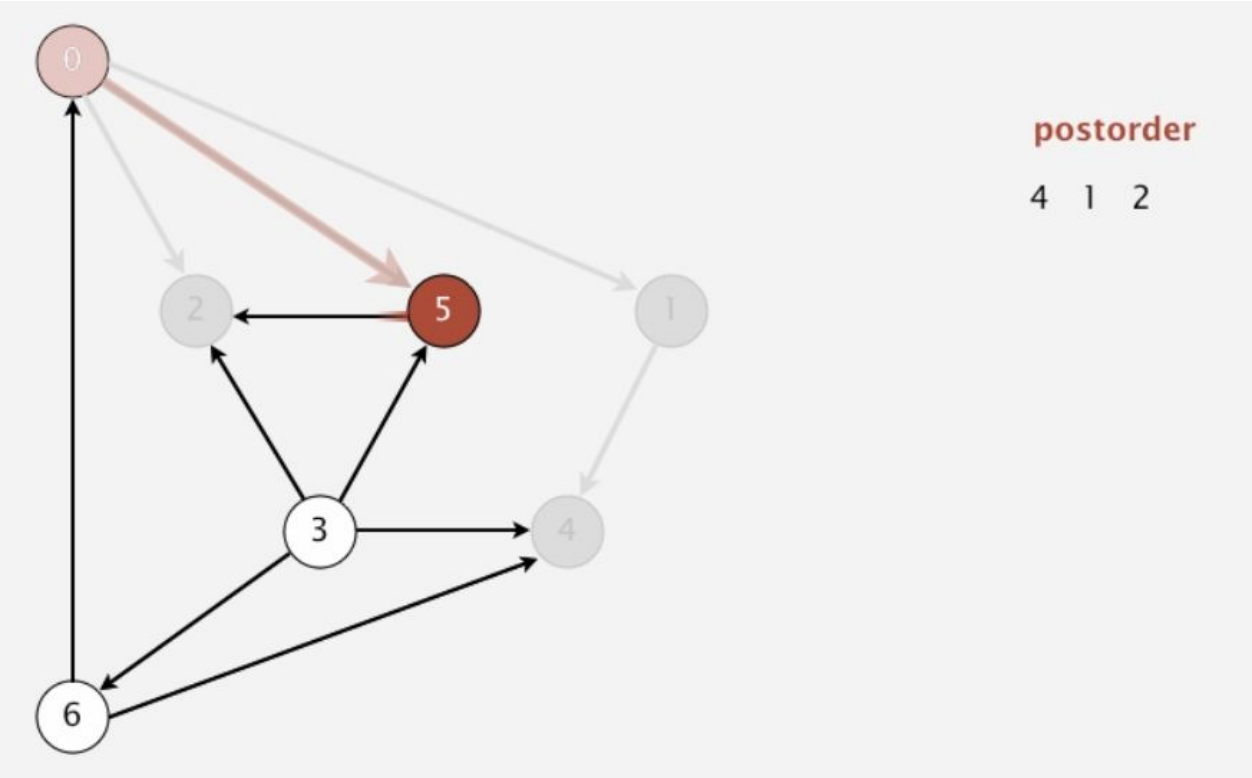
# Топологічне сортування

- Повернулися назад.
- З 0 є вихідне ребро.
- Відвідати 2. з два немає вихідних ребер, додати в postorder



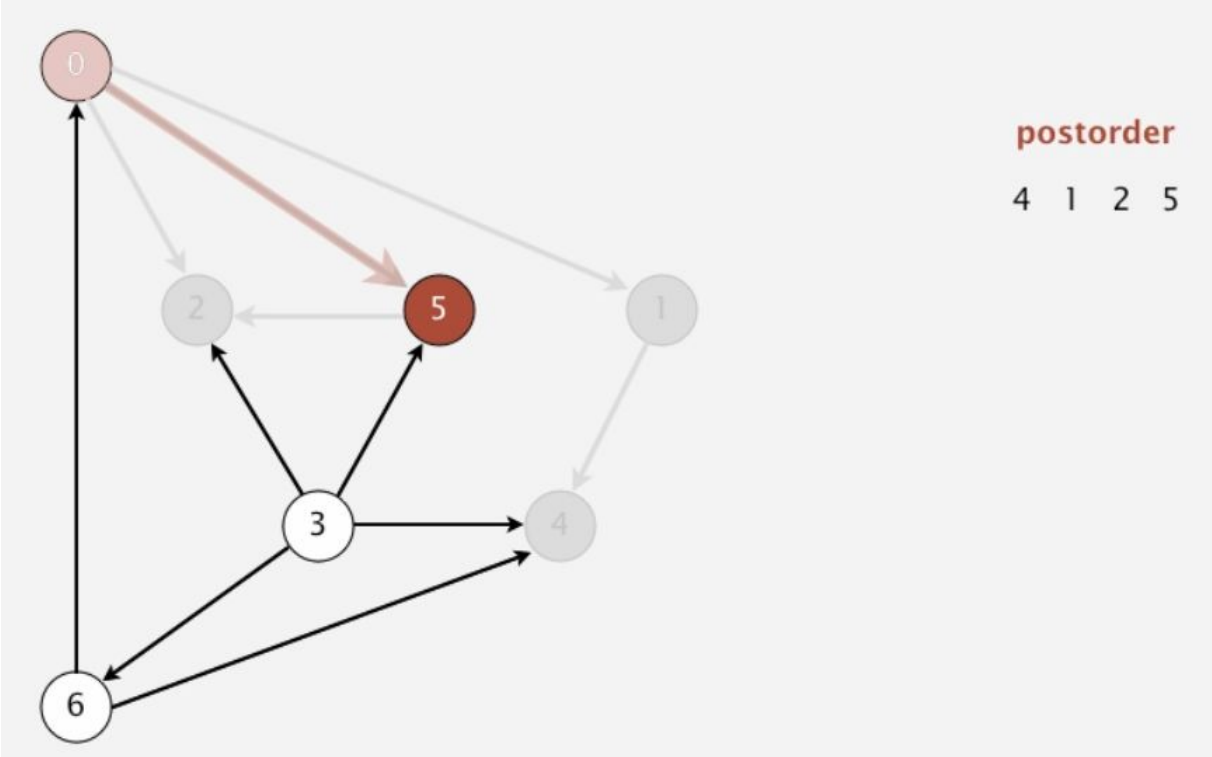
# Топологічне сортування

- Повернулися назад.
- З 0 є вихідне ребро.
- Відвідати 5. З 5 є вихідне ребро.



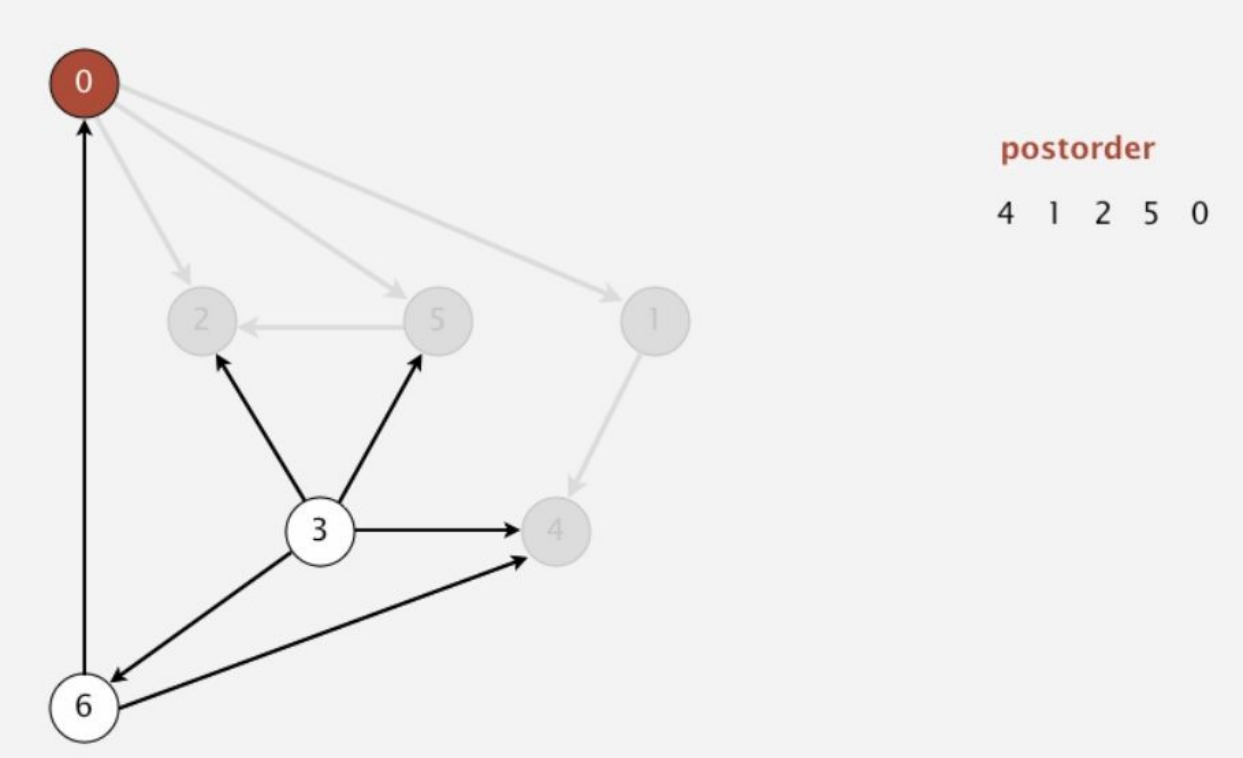
# Топологічне сортування

- 2 вже відвідане.
- З 5 більше немає вихідних ребер.
- Повернути 5 і записати до postorder.



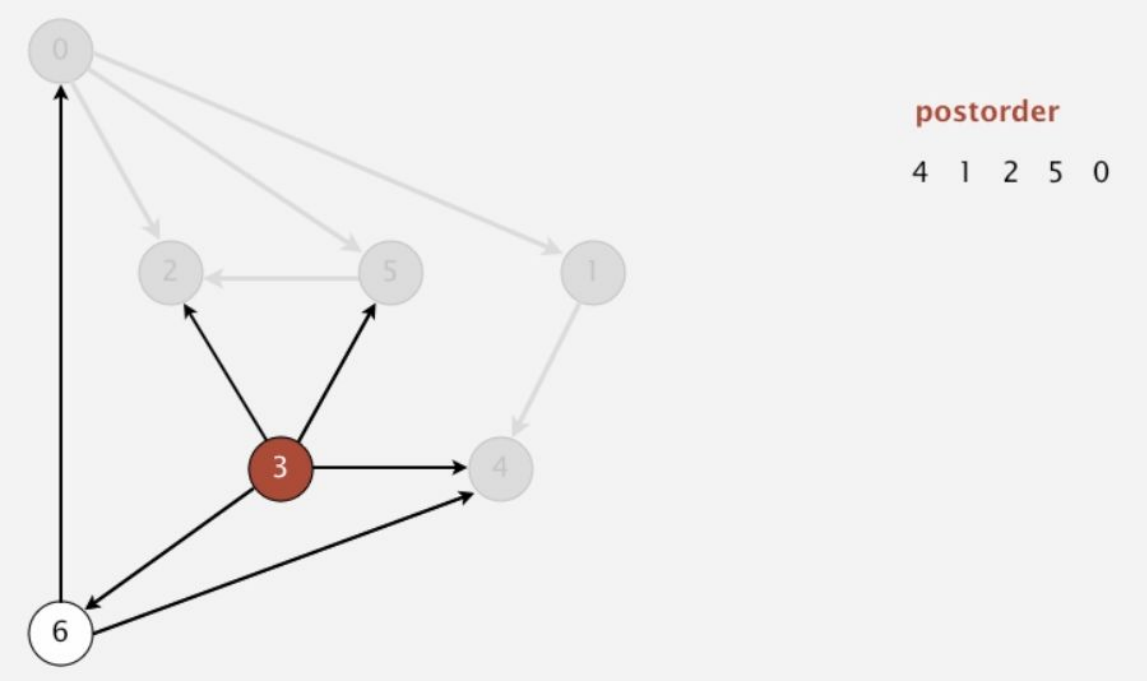
# Топологічне сортування

- Повернулися в 0.
- З 0 немає більше не відвіданих ребер.
- Повернути 0.



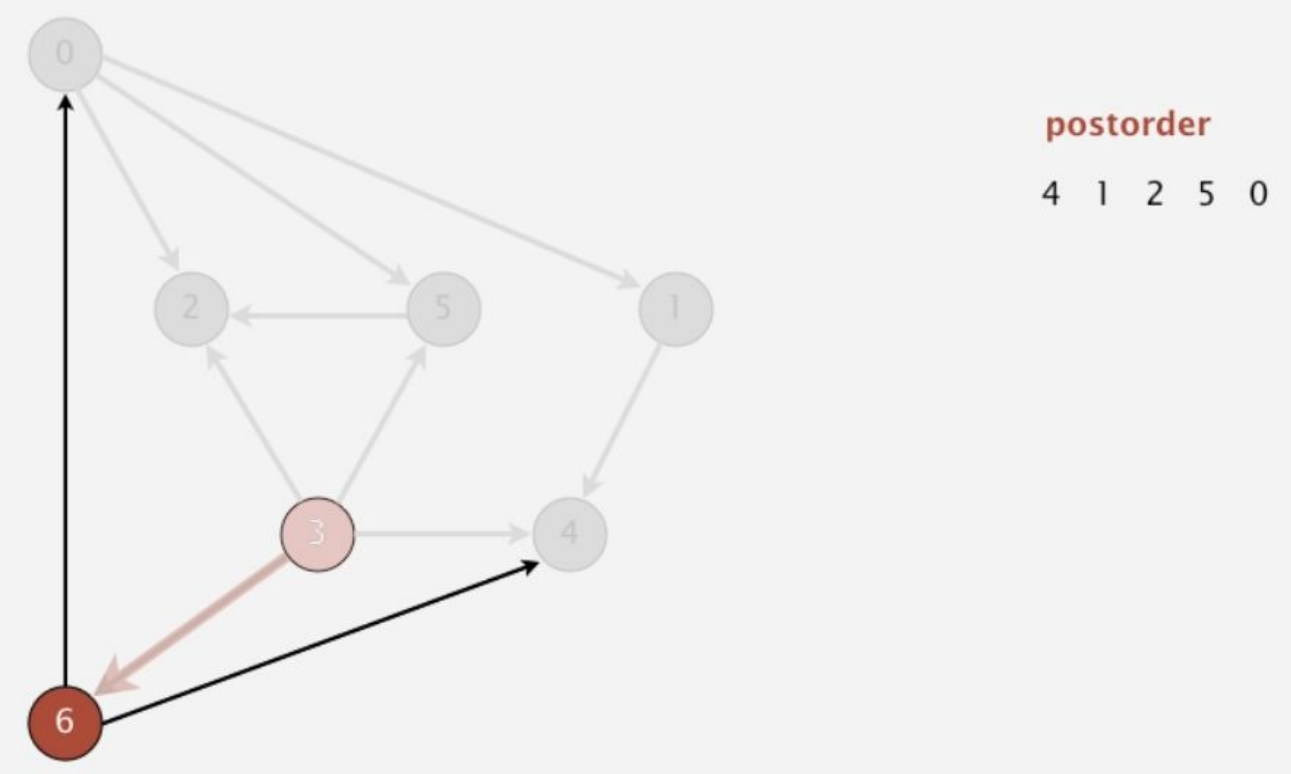
# Топологічне сортування

- 0 відвідане.
- Переглядаємо наступні вузли 1,2 (вони обидва вже відвідані.
- Наступний не відвіданий вузол 3.
- Відвідати 3.



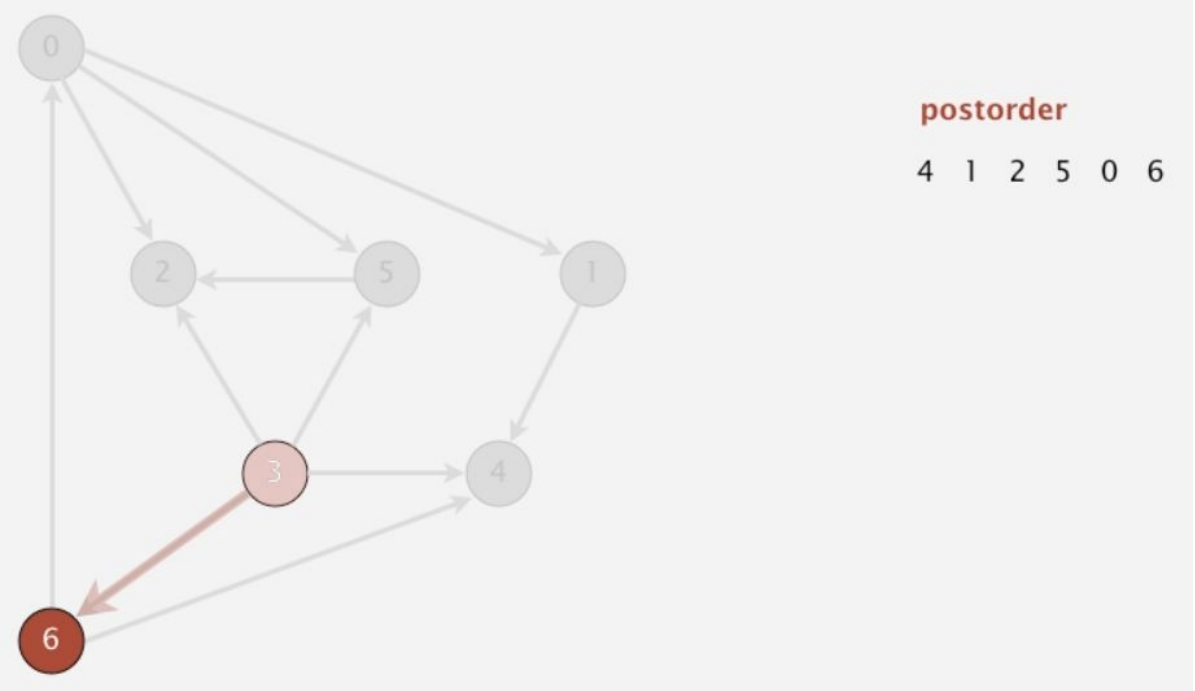
# Топологічне сортування

- Відвідати 2, відвідано.
- Відвідати 4, відвідано.
- Відвідати 5, відвідано.
- Відвідати 6.



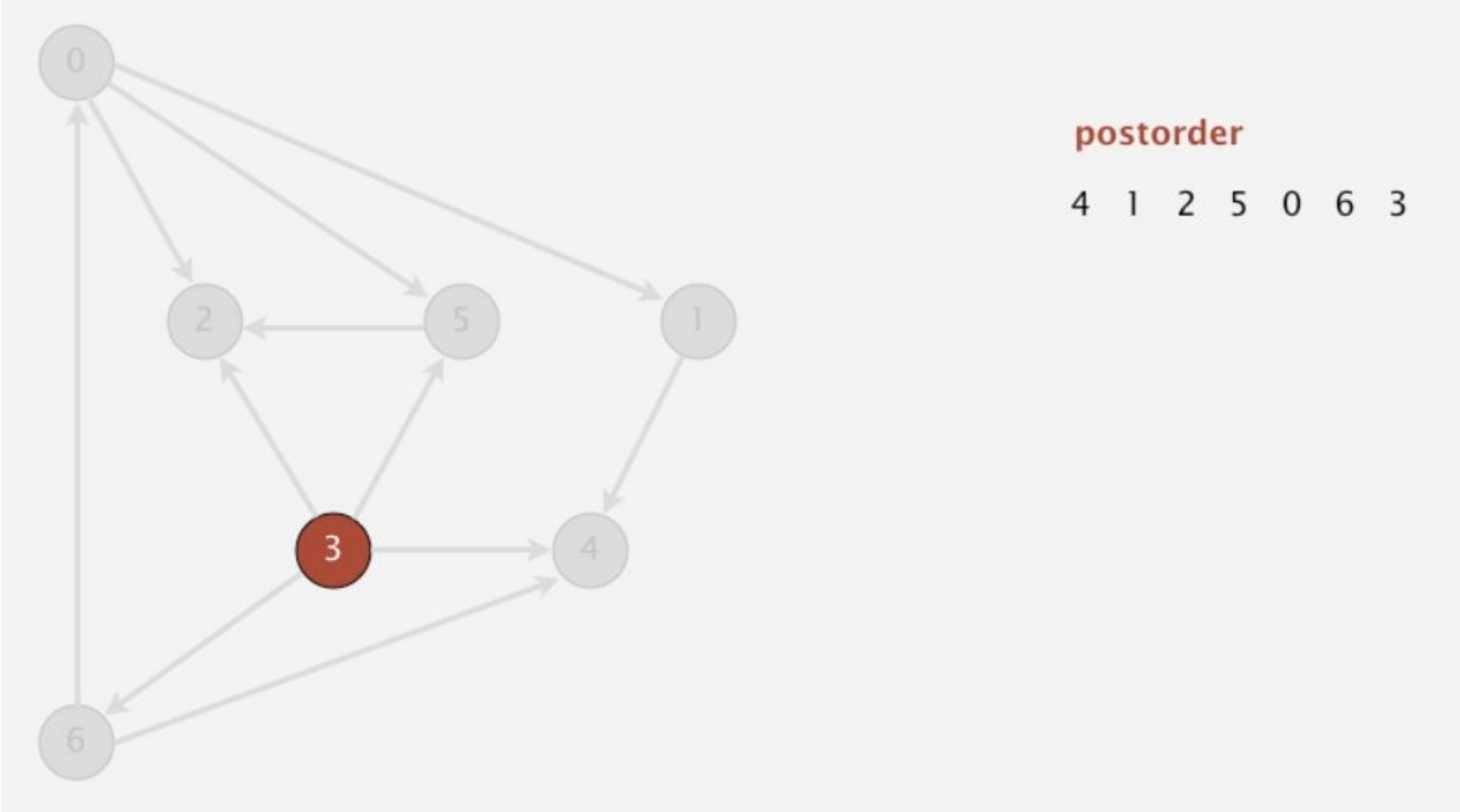
# Топологічне сортування

- З 6 є вихідні ребра.
- 0 відвідане
- 4 відвідане
- Повернути 6 і записати в postorder



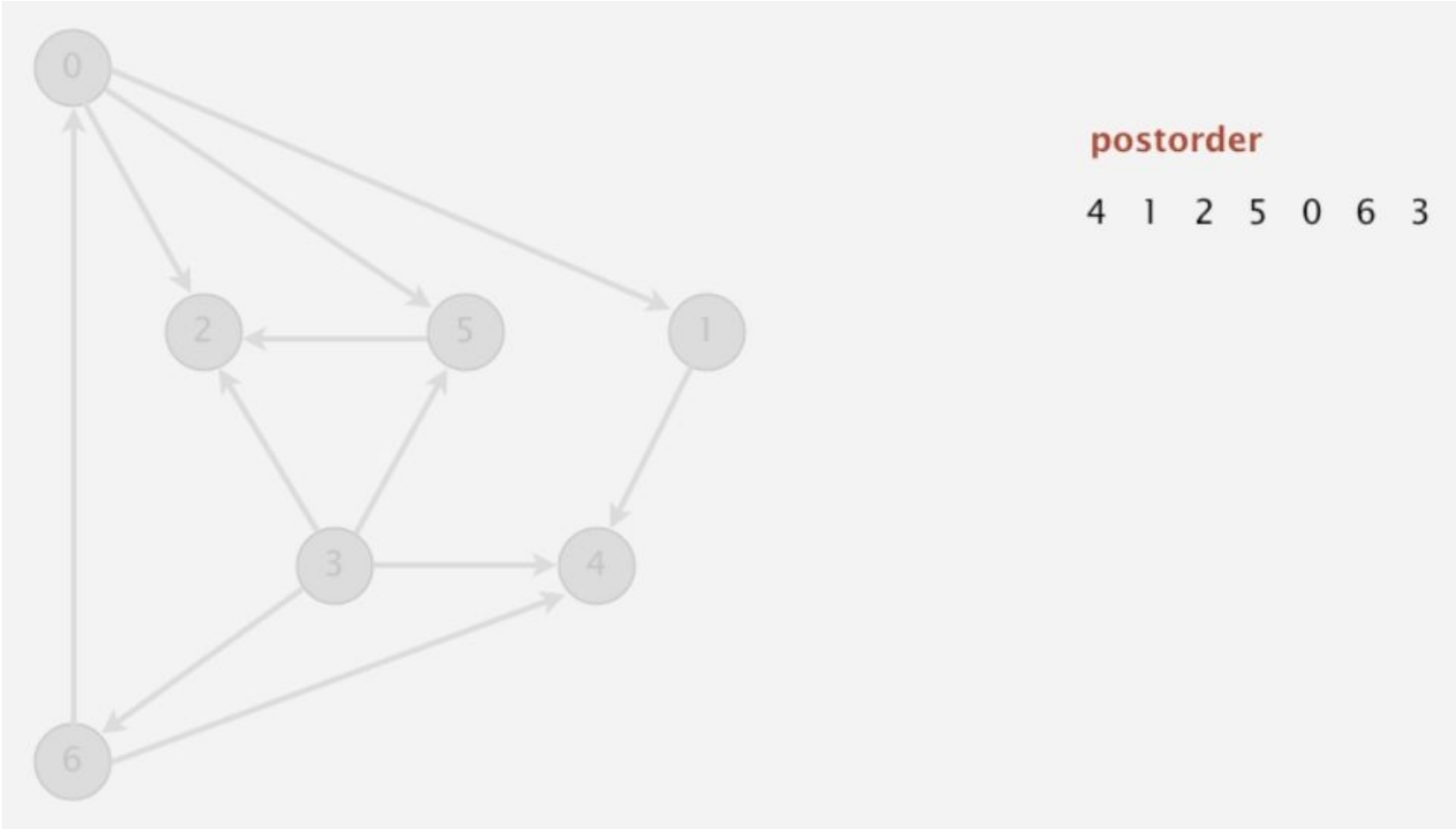
# Топологічне сортування

- З 3 більше немає ребер, повернути 3



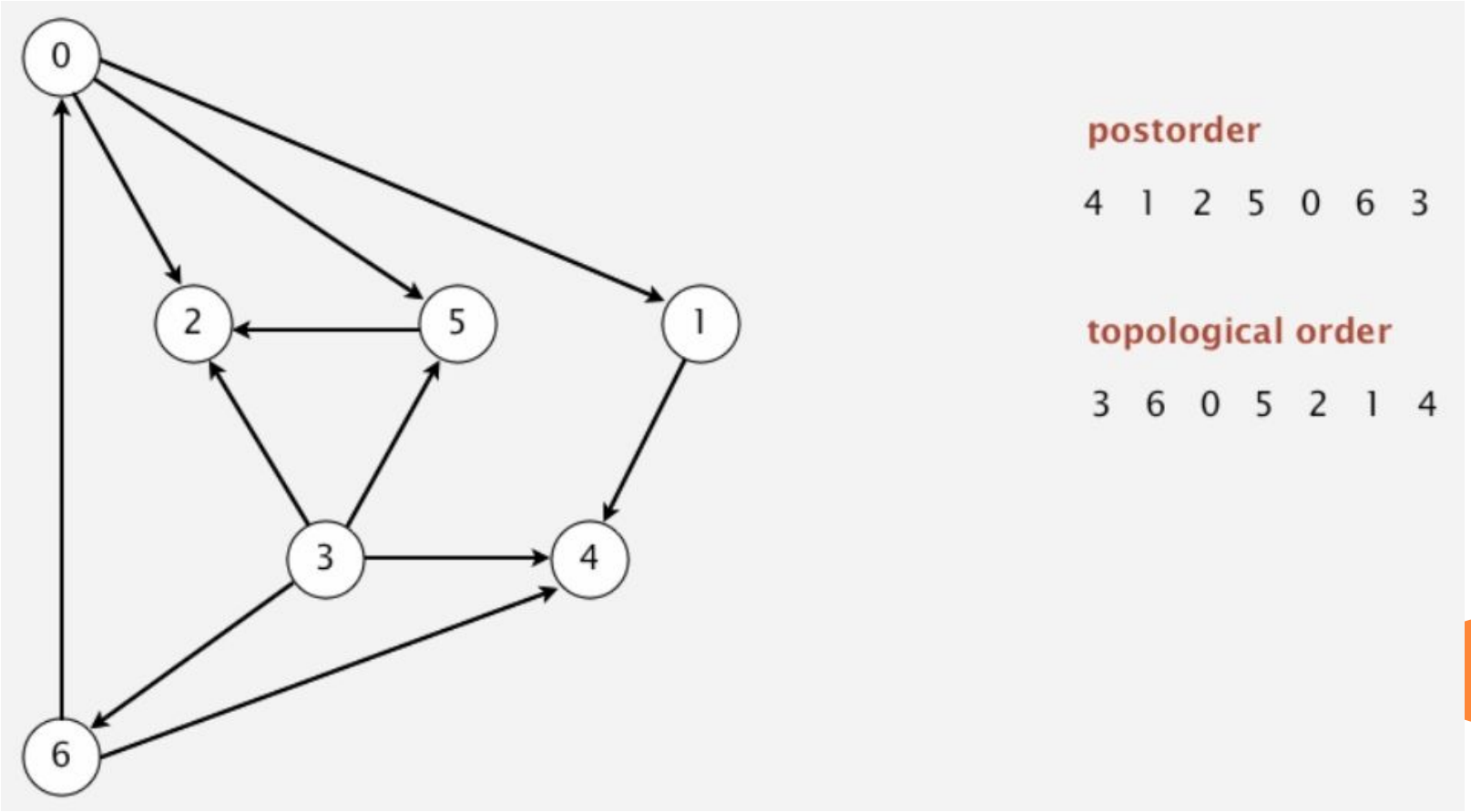
# Топологічне сортування

- Всі вузли відвідані. Ми отримали postorder.



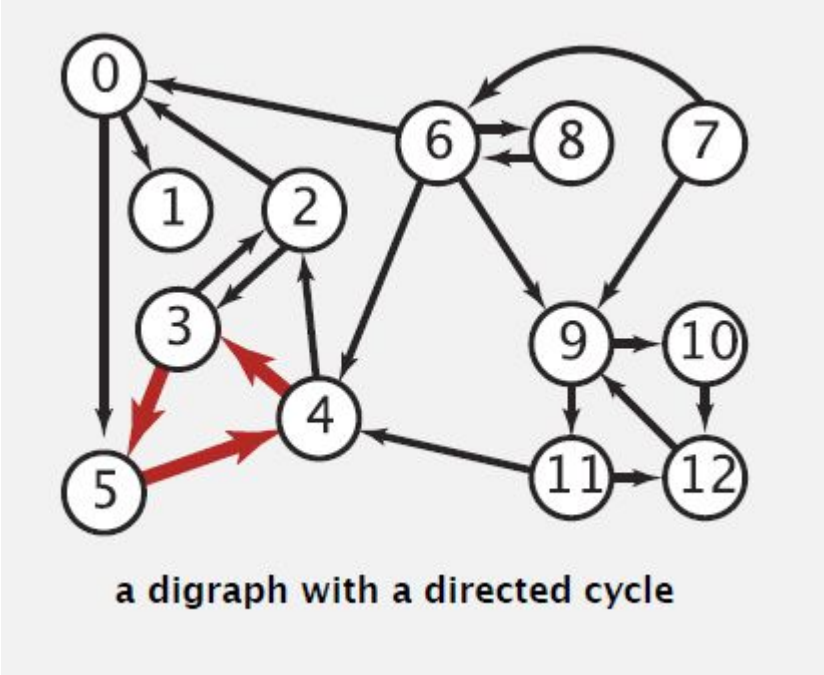
# Топологічне сортування

- Перевертаємо postorder і отримаємо топологічну чергу.



# Топологічне сортування

- Орграф має топологічну чергу, якщо не має циклів.
- Досить просто вирішуєма задача, пошуку циклів в орієнтованому графі.



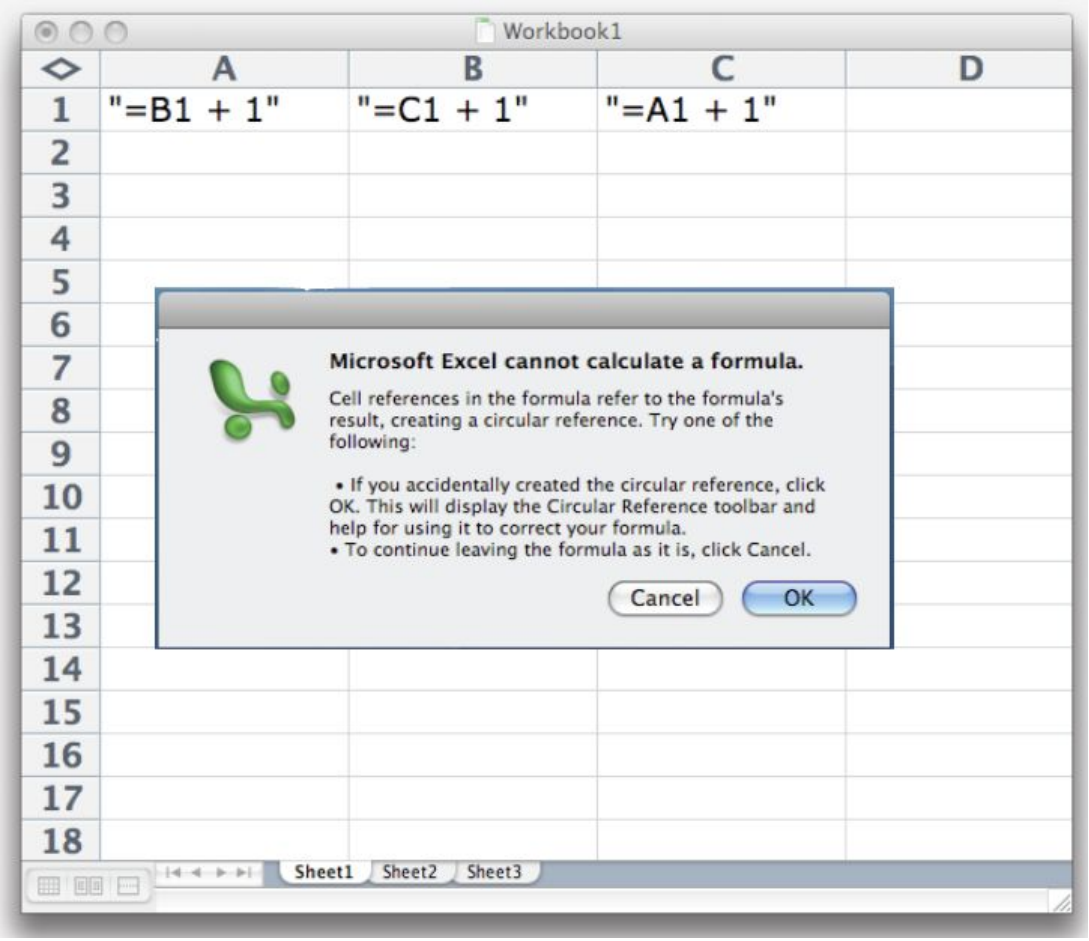

a digraph with a directed cycle

# Цикли в орієнтовних графах

- Компілятор Java ідентифікує цикли.
- Спробуйте:
  - public class A extends B{
    - ...
  - }
  - public class B extends C{
    - ...
  - }
  - public class C extends A{
    - ...
  - }
  - компілятор видасть помилку.

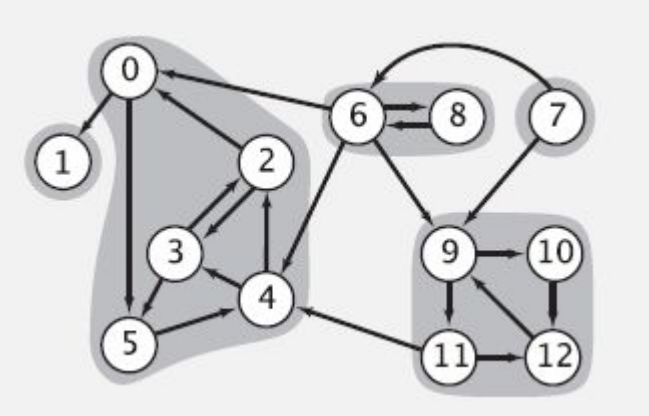
# Цикли в орієнтовних графах

- Microsoft Excell робить пошук циклів



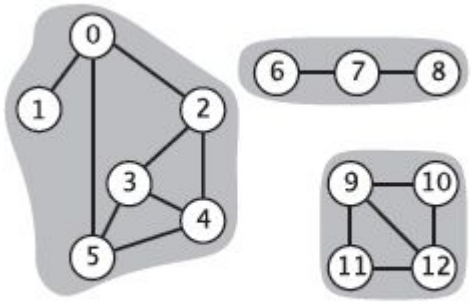
# Сильно зв’язані компоненти

- Вершини v і w є сильно зв’язаними (СЗ) якщо існує направлений шлях з ***v*** в ***w*** і з ***w*** в ***v***.
- Основні властивості:
  - vсильно зв’язана з v
  - якщо v СЗ з w тоді і w СЗ з v
  - якщо v СЗ з w, а w СЗ з x, тоді v СЗ з x
- Сильно зв’язаним компонентом називають максимальну підмножину сильно зв’язаних вершин



# Сильно зв’язані компоненти

- Згадаємо спочатку неорієнтовані графи і зв’язані компоненти.
- На малюнку 3 з’єднаних компоненти
- Як ми пам’ятаємо досить просто обрахувати id компонента за допомогою DFS



3 connected components

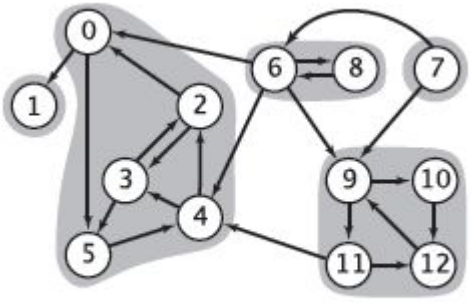
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| cc[] | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 2 | 2 |

- public int connected(int v, int w){ return cc[v] == cc[w];
- }

# Сильно зв’язані компоненти

- З орієнтованими графами в нас з’являється умова сильної зв’язаності вершин.
- На малюнку 5 СЗК.
- Ми знову можемо використати масив для зберігання СЗК id

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| scc[] | 1 | 0 | 1 | 1 | 1 | 1 | 3 | 4 | 3 | 2 | 2 | 2 | 2 |



5 strongly-connected components

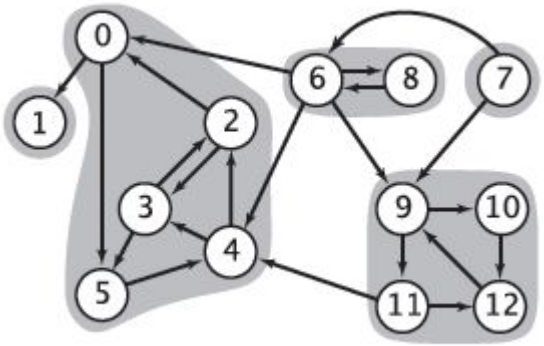
- public int stronglyConnected(int v, int w){
  - return scc[v] == scc[w];
- }

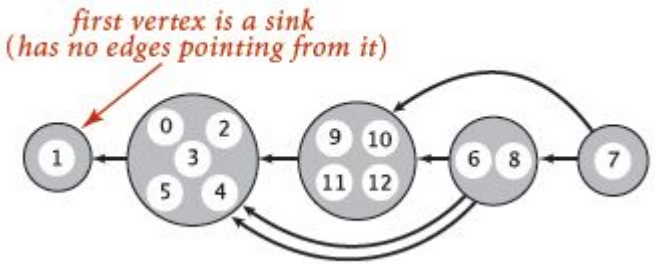
# Сильно зв’язані компоненти

- До 80-х років не існувало простих алгоритмів знаходження сильно зв’язаних компонентів.
- В 1978 р. з’явився простий двопрохідний алгоритм з лінійним часом, який приписують Косараджу.
- Хоча пізніше цей алгоритм був знайдений в російській літературі датованій 72 роком.
- В 90-х з’явився більш простий алгоритм Cheriyan-Mehlhorn.

# Алгоритм Косараджу

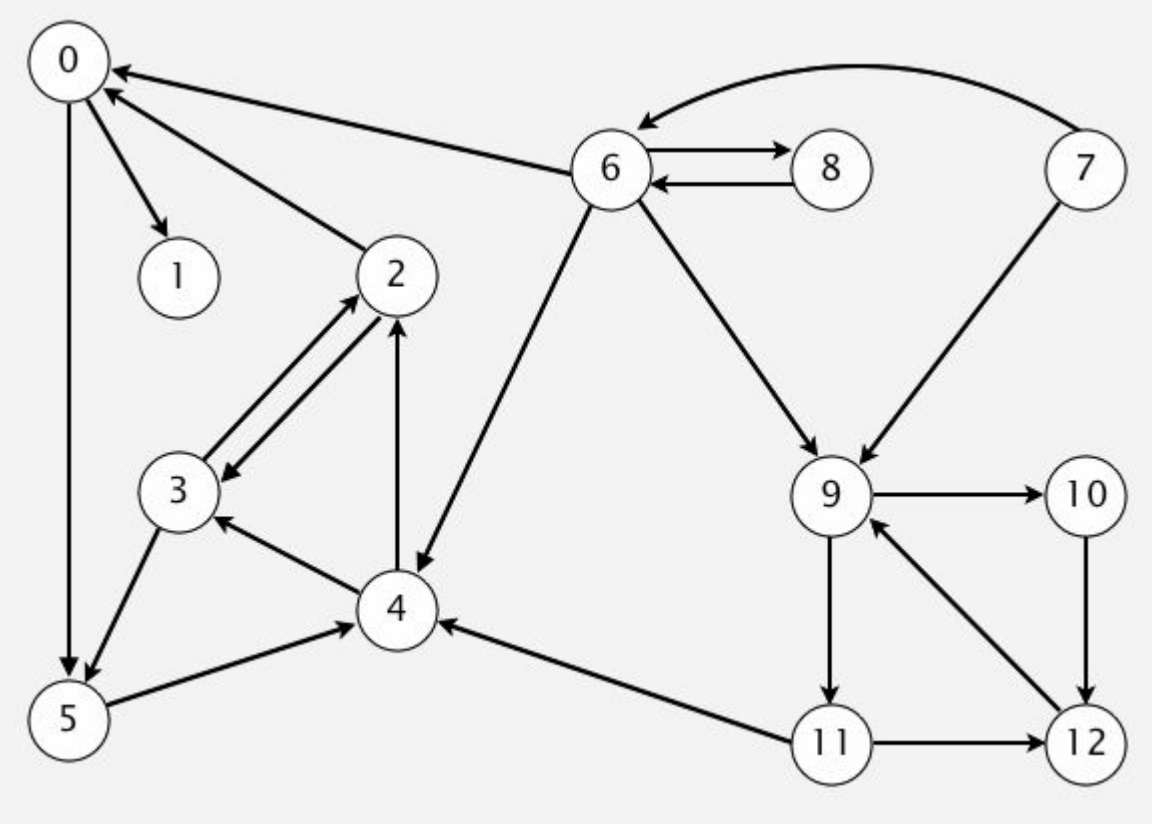
- В алгоритмі використовується той факт, що в транспонованому орграфі (той самий граф з оберненими напрямками ребер) має ті самі сильно зв’язані компоненти, що й початковий граф.
- Основна ідея:
  - Обрахувати топологічну чергу в трансопонованому орграфі
  - Запустити DFS використовуючи топологічну чергу



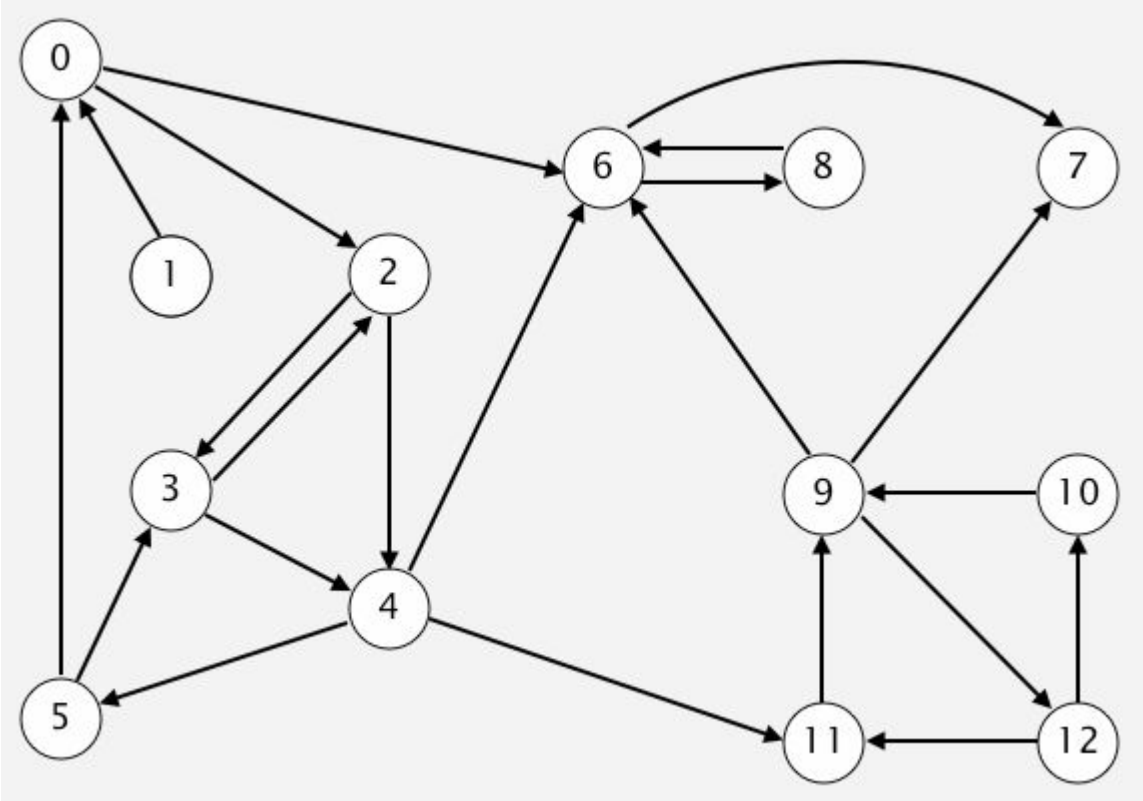
# Алгоритм Косараджу

- Почнемо з фази один. Обрахунок топологічної черги.



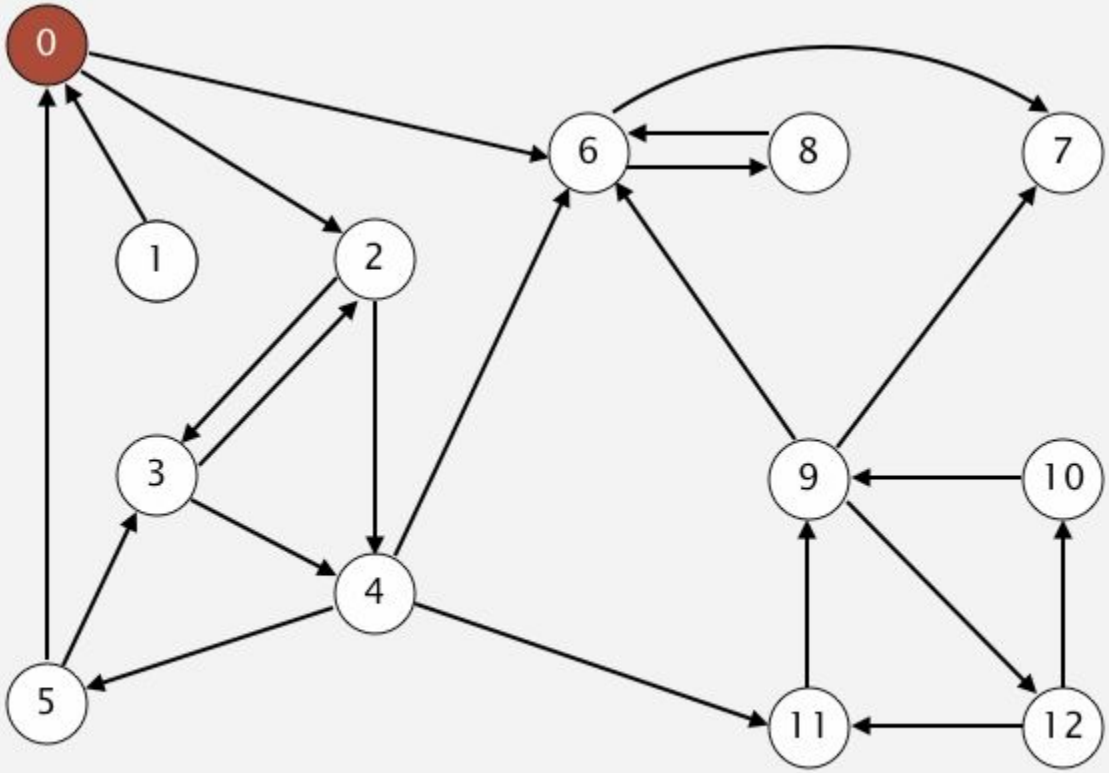
# Алгоритм Косараджу

- Отримаємо транспонований орграф



# Алгоритм Косараджу

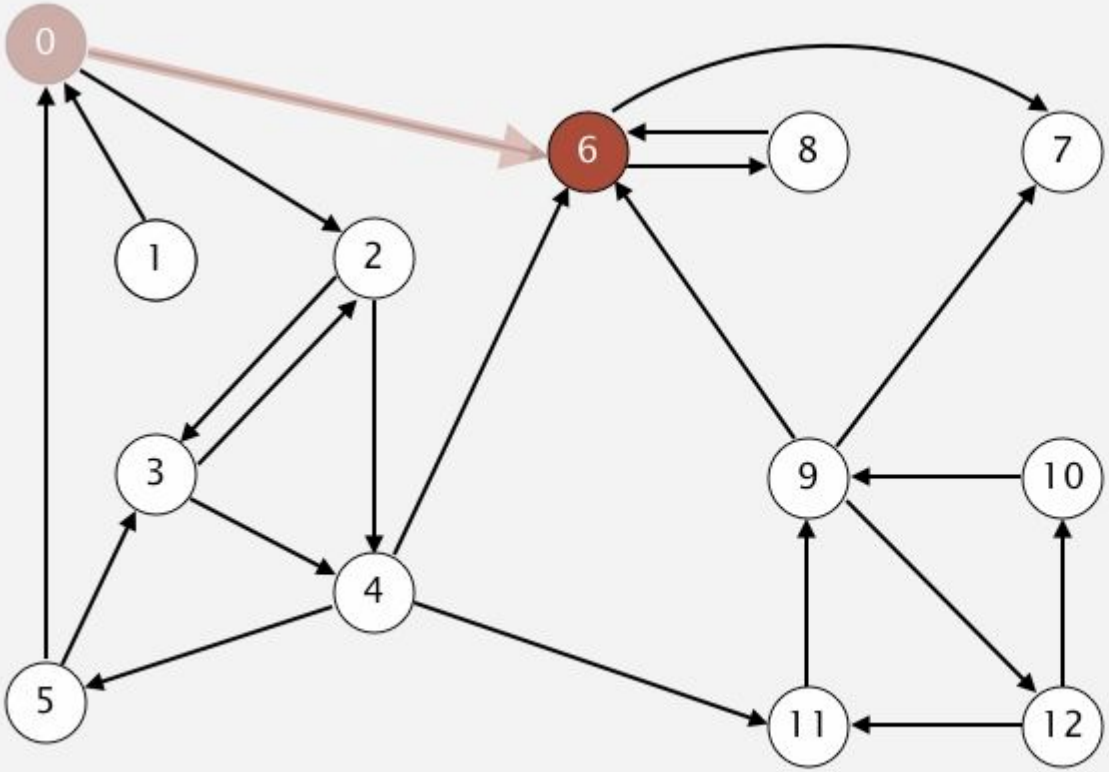
- Відвідати 0.




| v | marked[v] |
|---|---|
| 0 | T |
| 1 | F |
| 2 | F |
| 3 | F |
| 4 | F |
| 5 | F |
| 6 | F |
| 7 | F |
| 8 | F |
| 9 | F |
| 10 | F |
| 11 | F |
| 12 | F |

# Алгоритм Косараджу

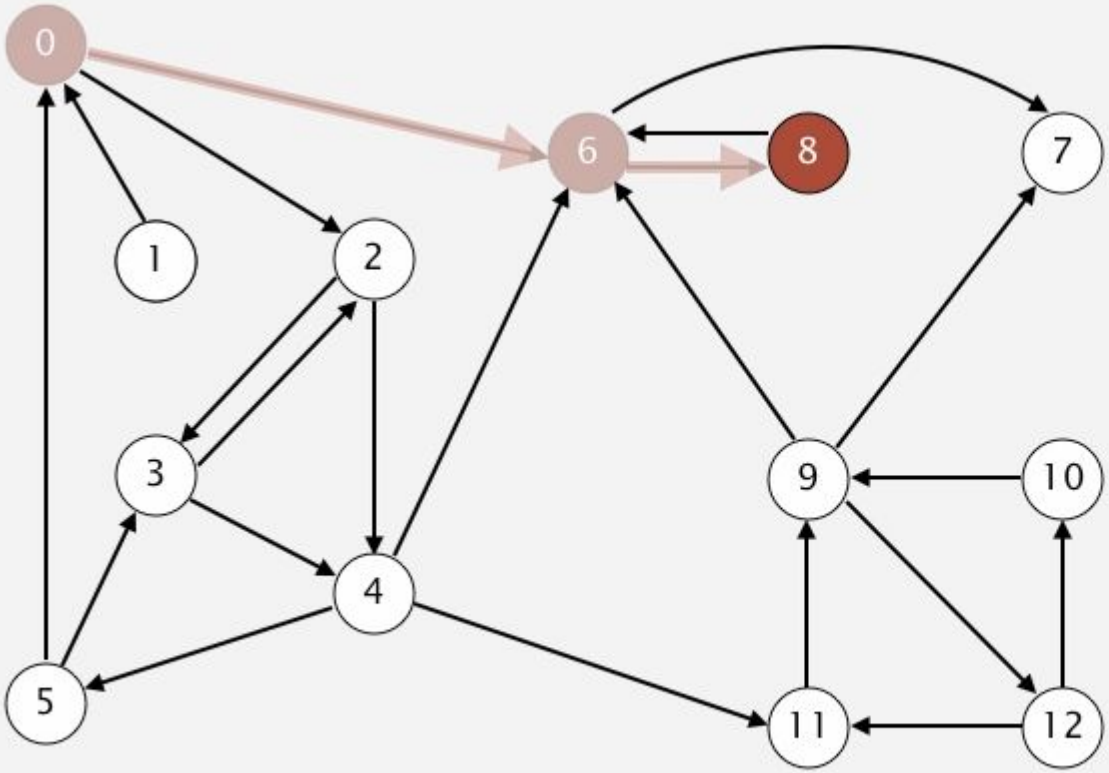
- Відвідати 6.




| v | marked[v] |
|---|---|
| 0 | T |
| 1 | F |
| 2 | F |
| 3 | F |
| 4 | F |
| 5 | F |
| 6 | T |
| 7 | F |
| 8 | F |
| 9 | F |
| 10 | F |
| 11 | F |
| 12 | F |

# Алгоритм Косараджу

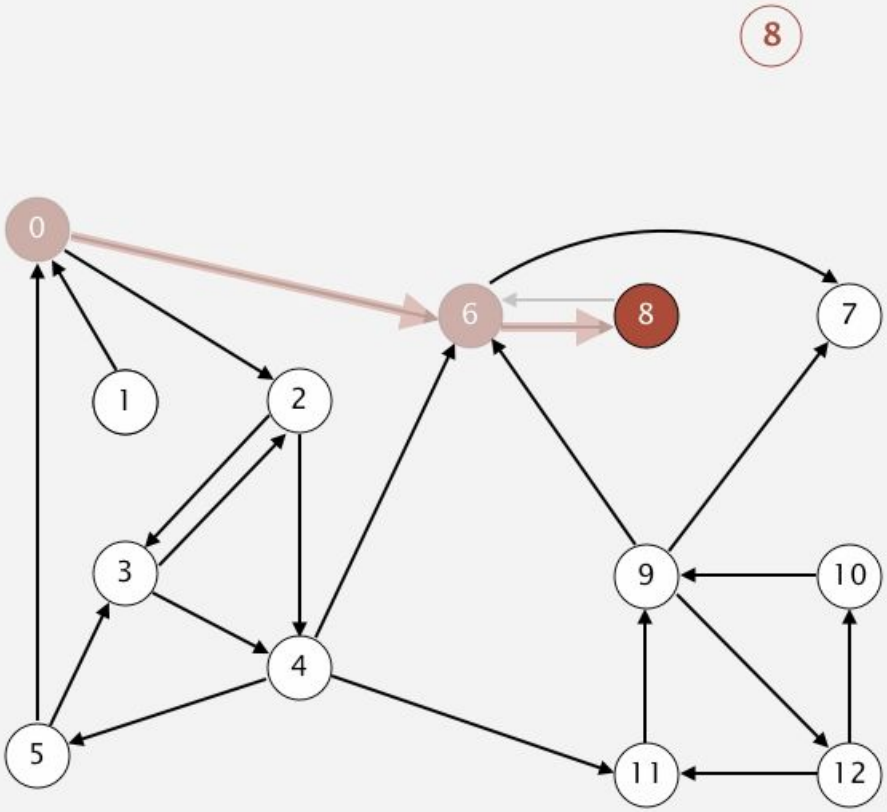
- Відвідати 8




| v | marked[v] |
|---|---|
| 0 | T |
| 1 | F |
| 2 | F |
| 3 | F |
| 4 | F |
| 5 | F |
| 6 | T |
| 7 | F |
| 8 | T |
| 9 | F |
| 10 | F |
| 11 | F |
| 12 | F |

# Алгоритм Косараджу

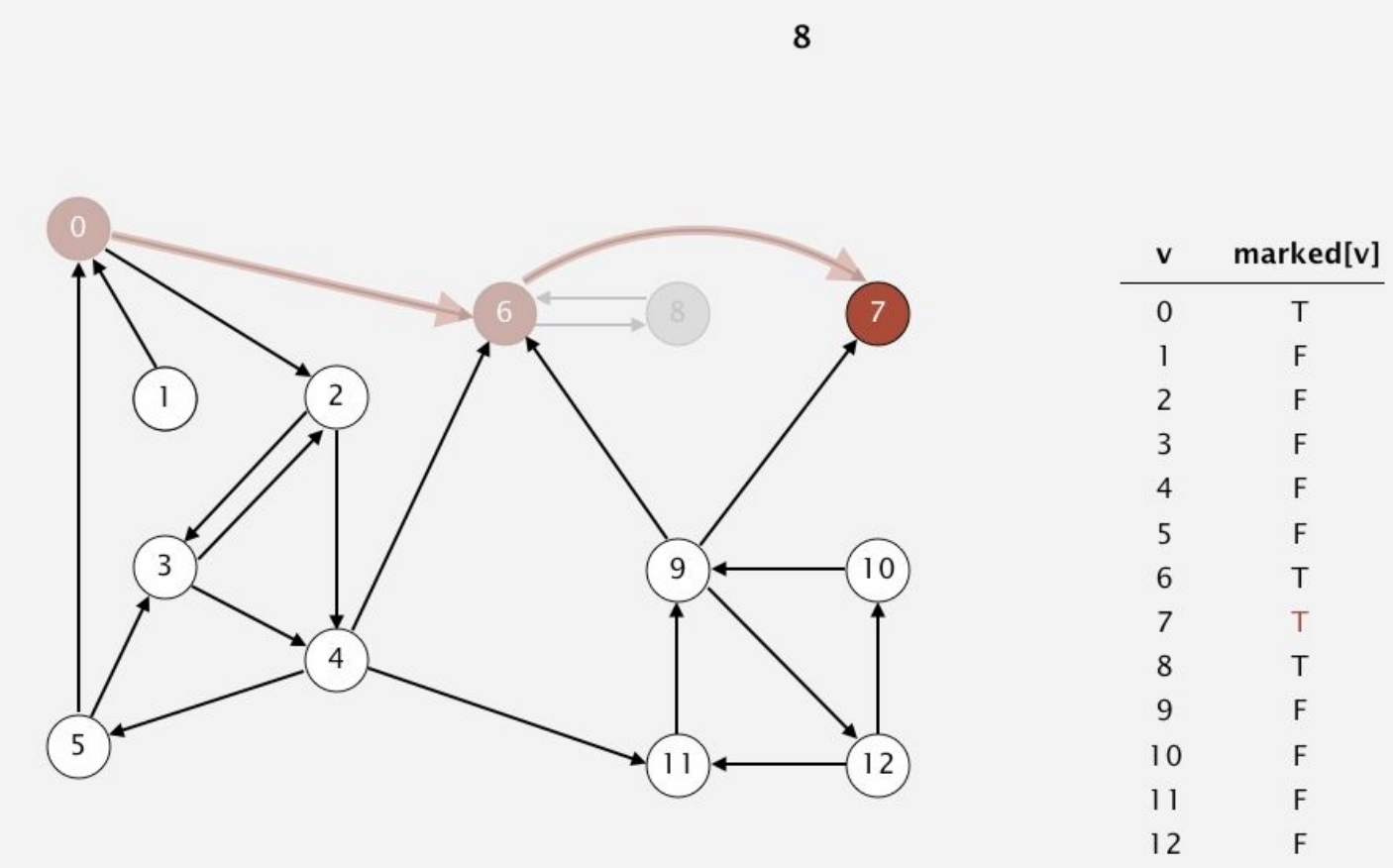
- Відвідати 6. Але 6 відвідано.
- Значить з 8 закінчили. Додати 8 до стеку.




| v | marked[v] |
|---|---|
| 0 | T |
| 1 | F |
| 2 | F |
| 3 | F |
| 4 | F |
| 5 | F |
| 6 | T |
| 7 | F |
| 8 | T |
| 9 | F |
| 10 | F |
| 11 | F |
| 12 | F |

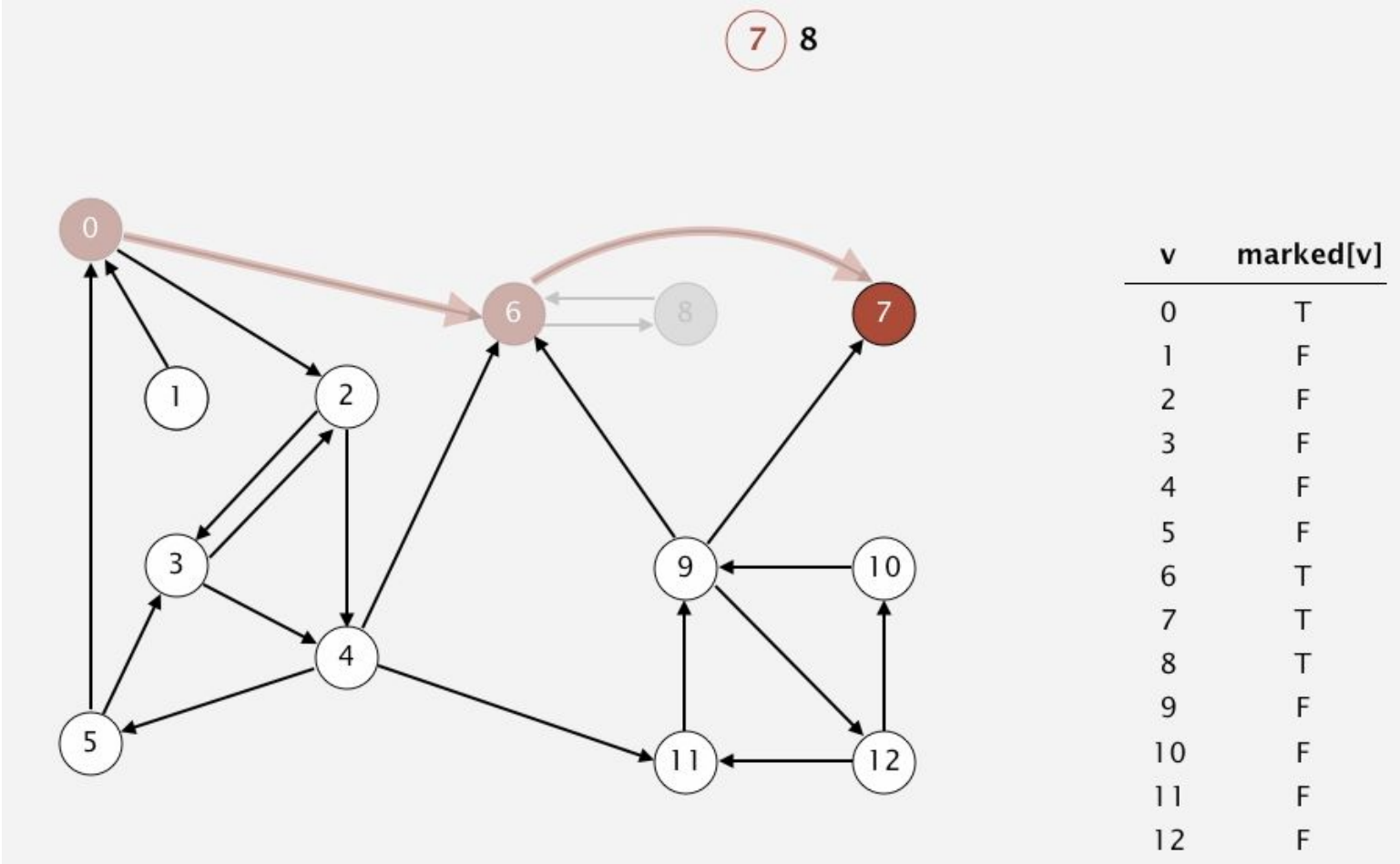
# Алгоритм Косараджу

- З 6 є перехід в 7.
- Відвідати 7.



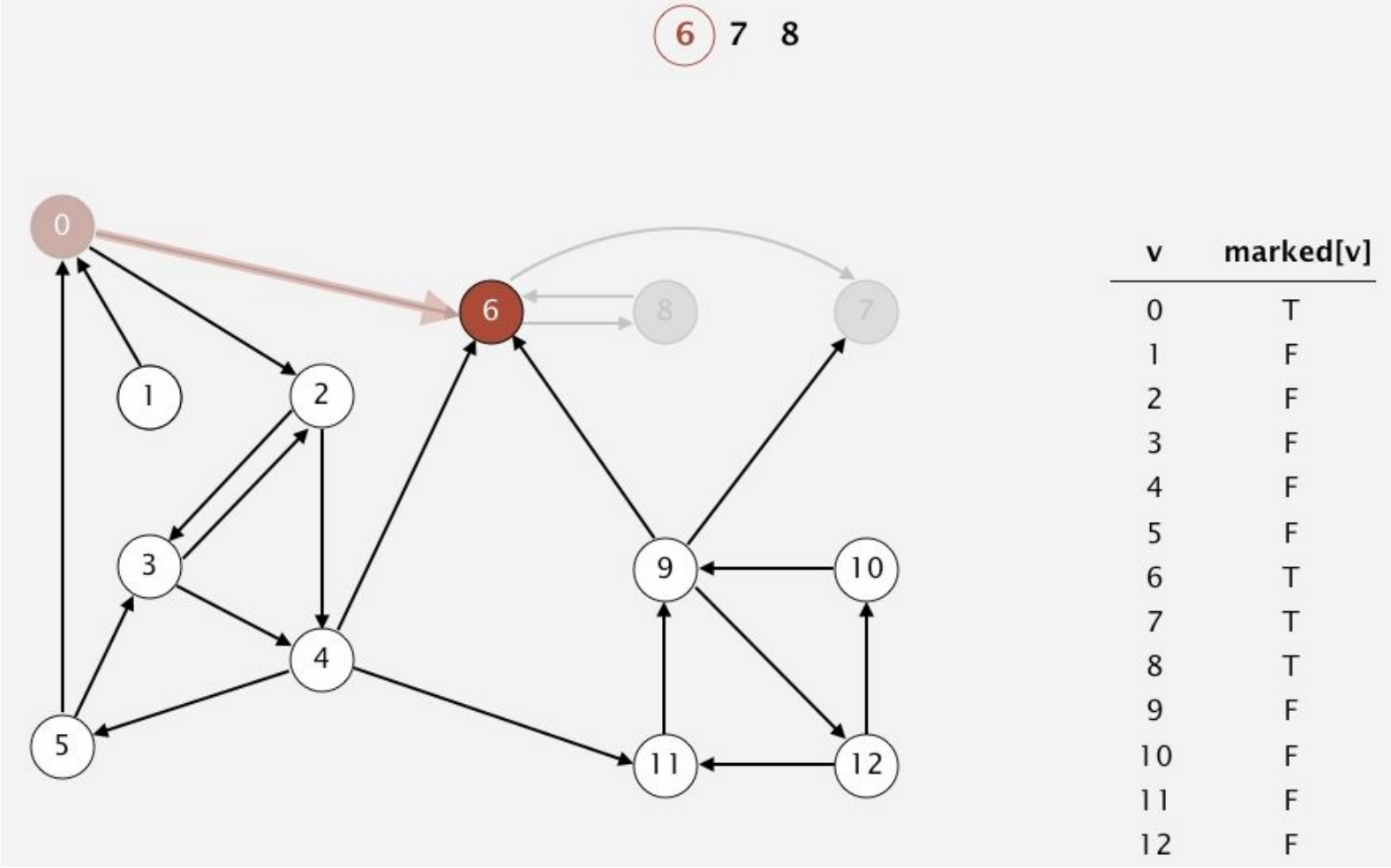
# Алгоритм Косараджу

- З 7 немає більше ребер. Додати 7 до стеку.



# Алгоритм Косараджу

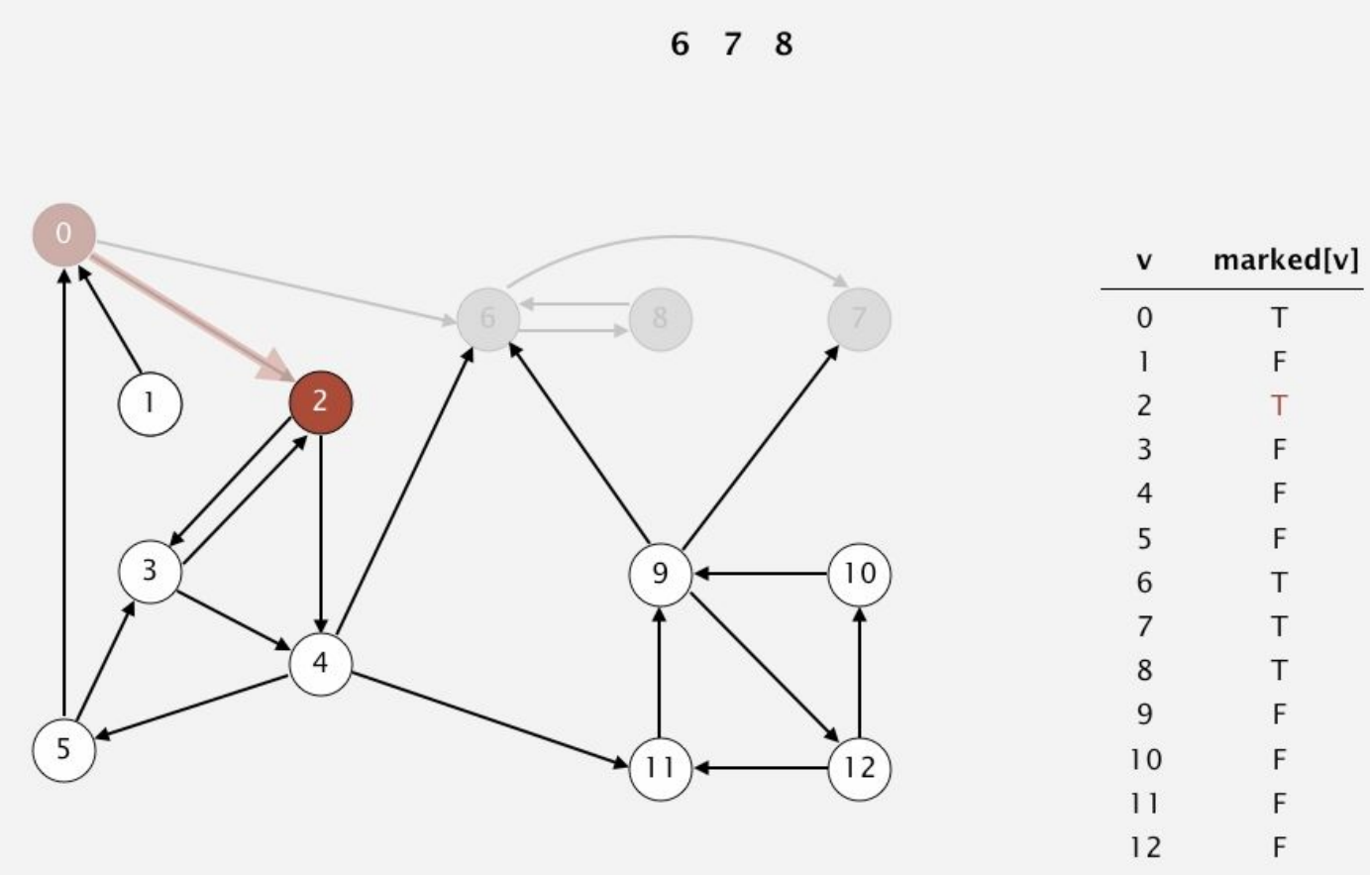
- З 6 немає більше ребер, додати 6 до списку.




| v | marked[v] |
|---|---|
| 0 | T |
| 1 | F |
| 2 | F |
| 3 | F |
| 4 | F |
| 5 | F |
| 6 | T |
| 7 | T |
| 8 | T |
| 9 | F |
| 10 | F |
| 11 | F |
| 12 | F |

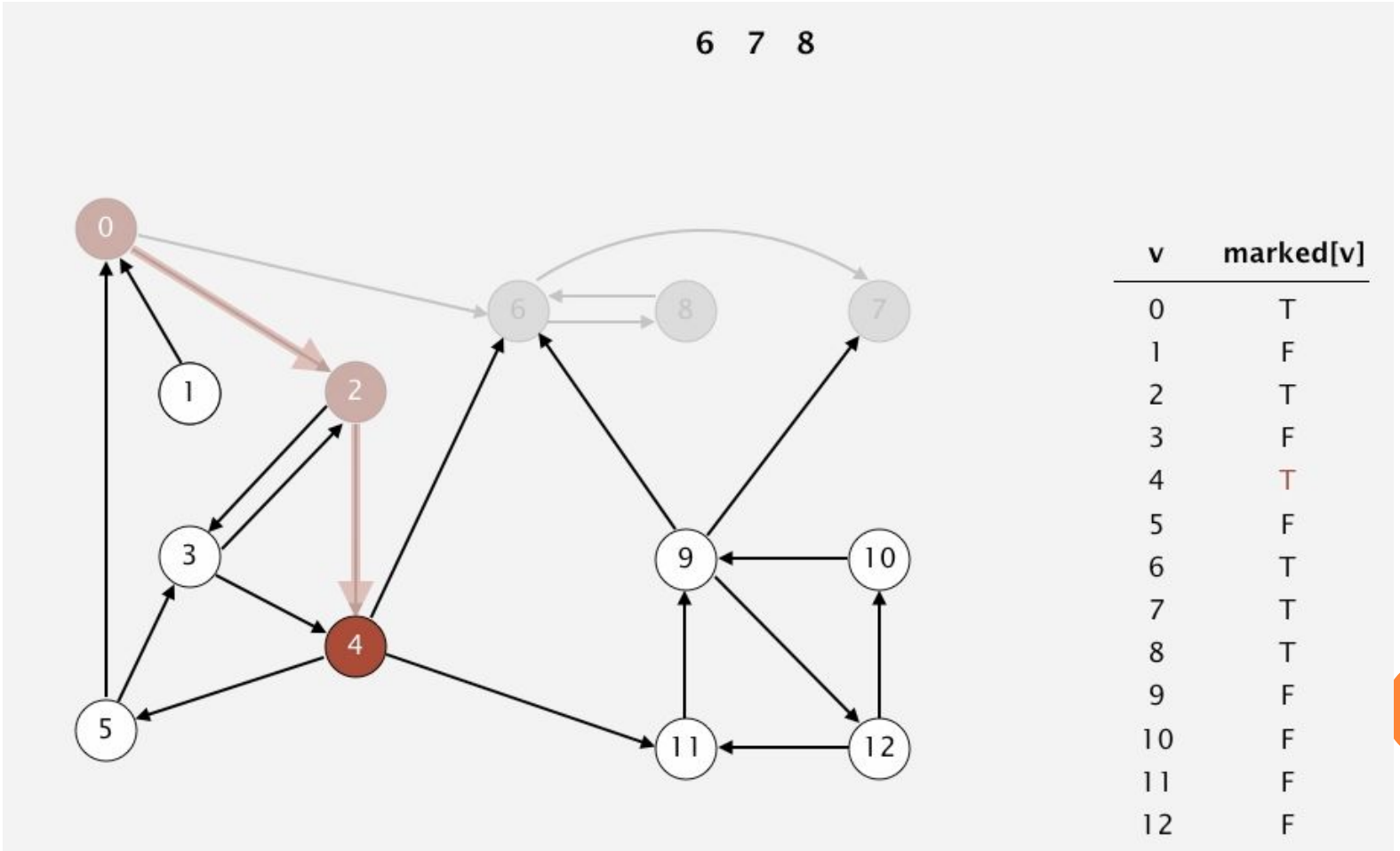
# Алгоритм Косараджу

- З 0 є ребро в 2.
- Відвідати 2.



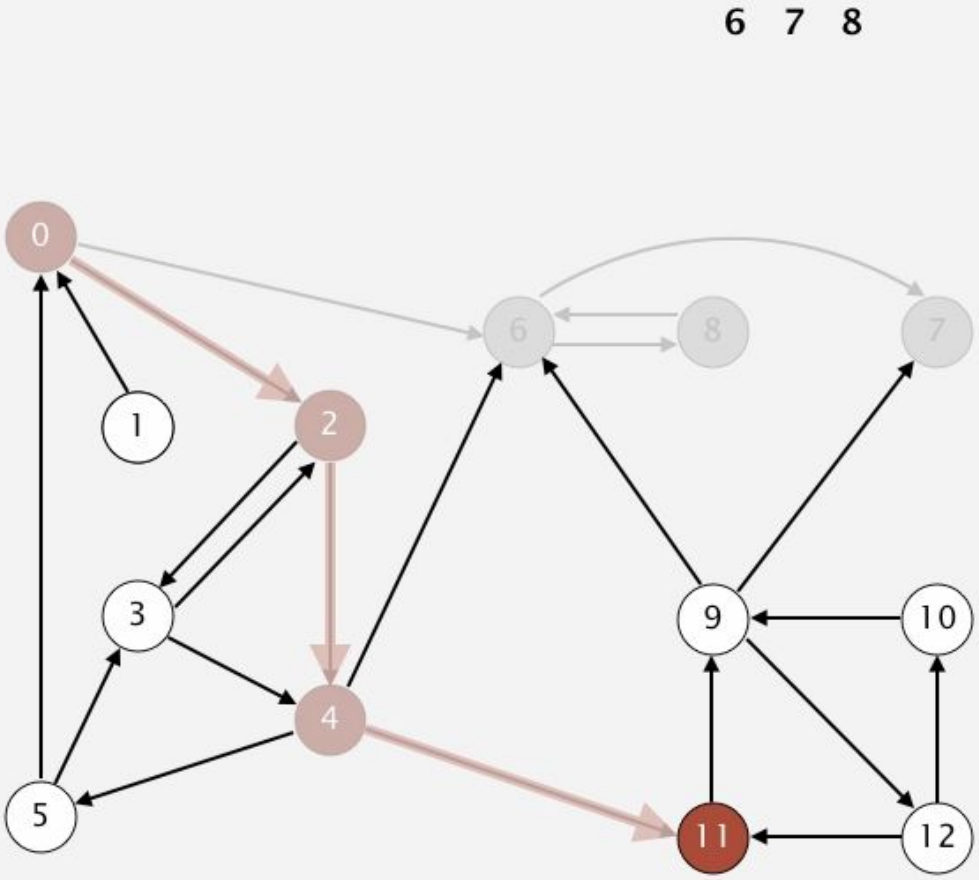
# Алгоритм Косараджу

- Відвідати 4.



# Алгоритм Косараджу

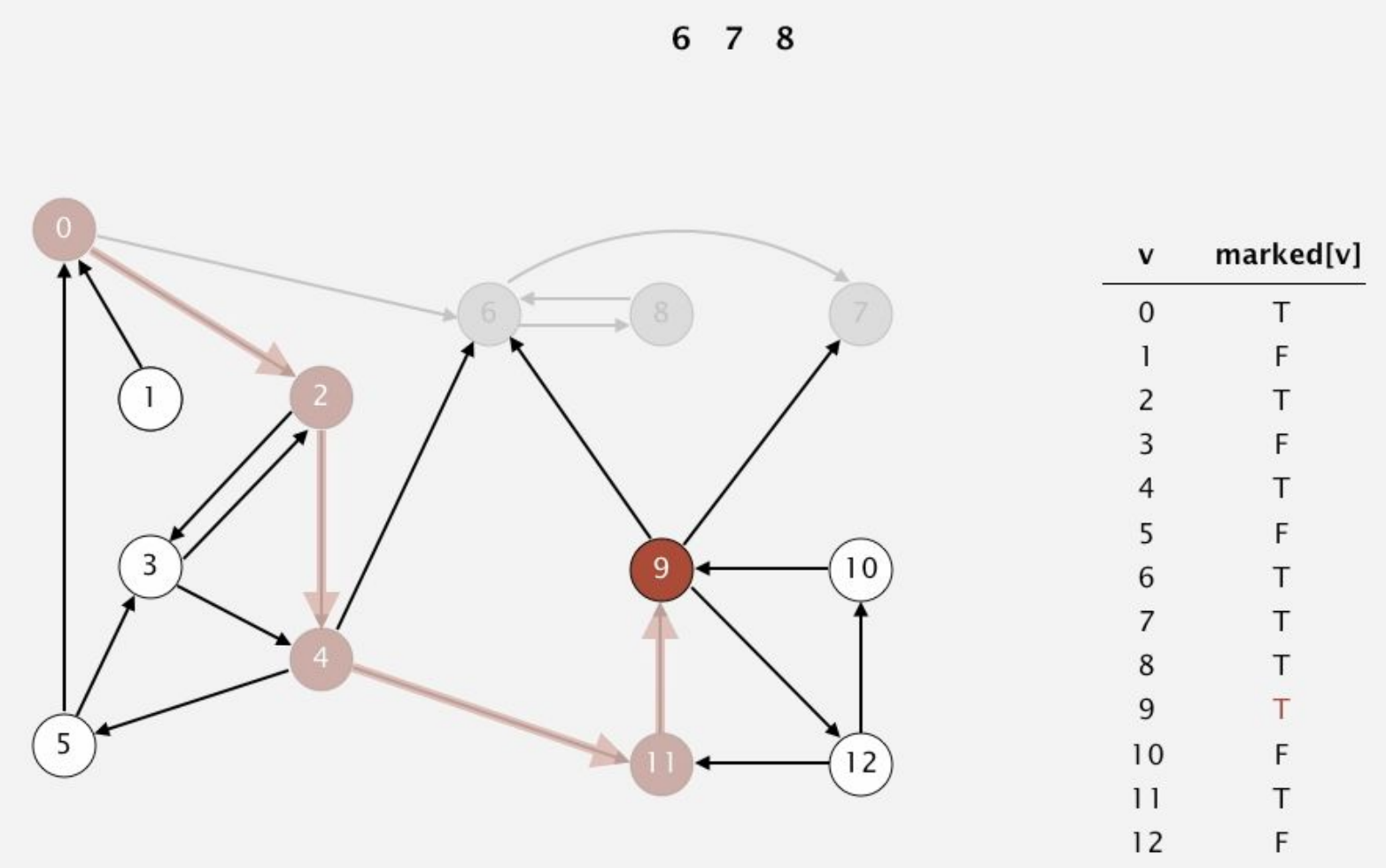
- Відвідати 11.




| v | marked[v] |
|---|---|
| 0 | T |
| 1 | F |
| 2 | T |
| 3 | F |
| 4 | T |
| 5 | F |
| 6 | T |
| 7 | T |
| 8 | T |
| 9 | F |
| 10 | F |
| 11 | T |
| 12 | F |

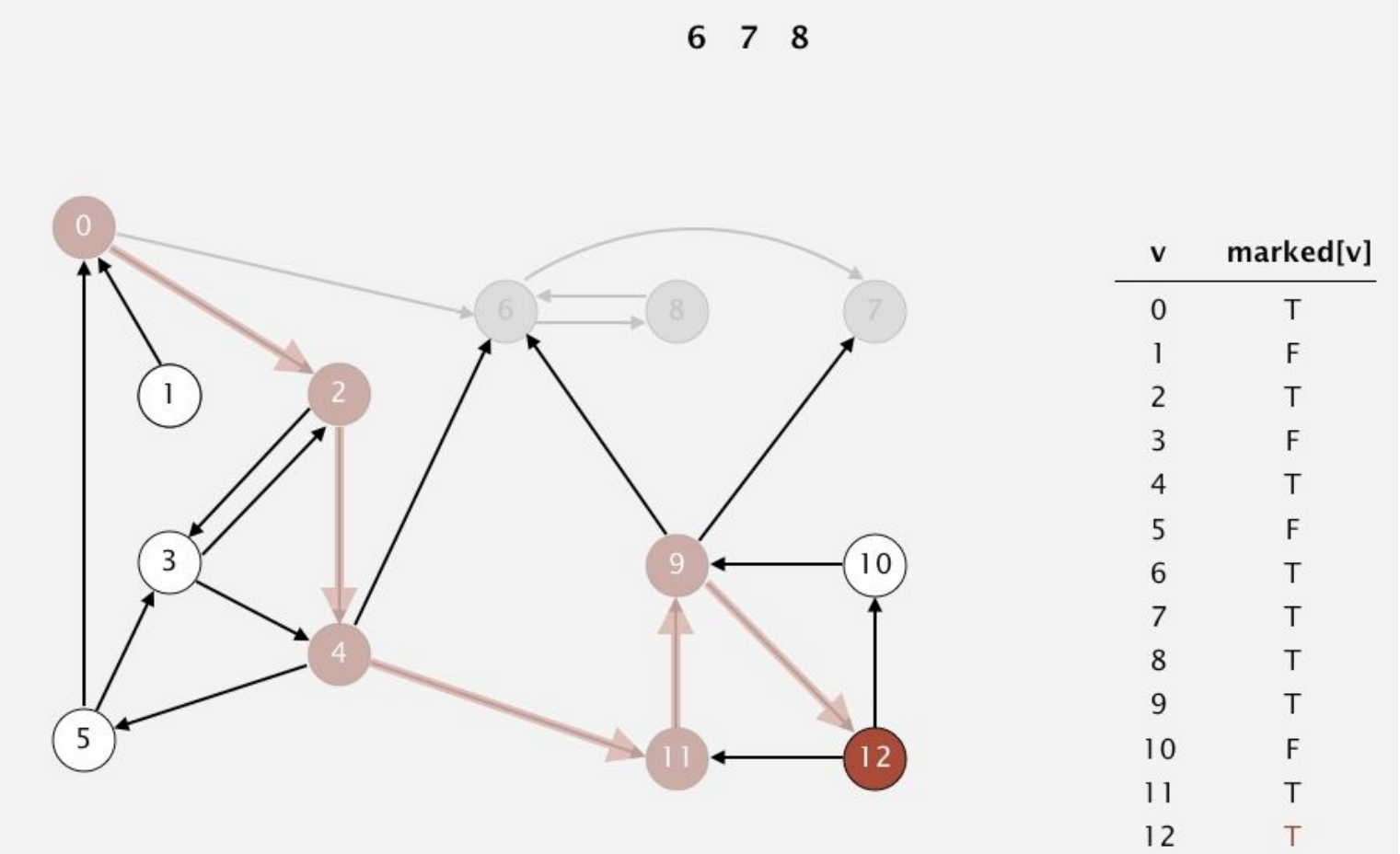
# Алгоритм Косараджу

- Відвідати 9.



# Алгоритм Косараджу

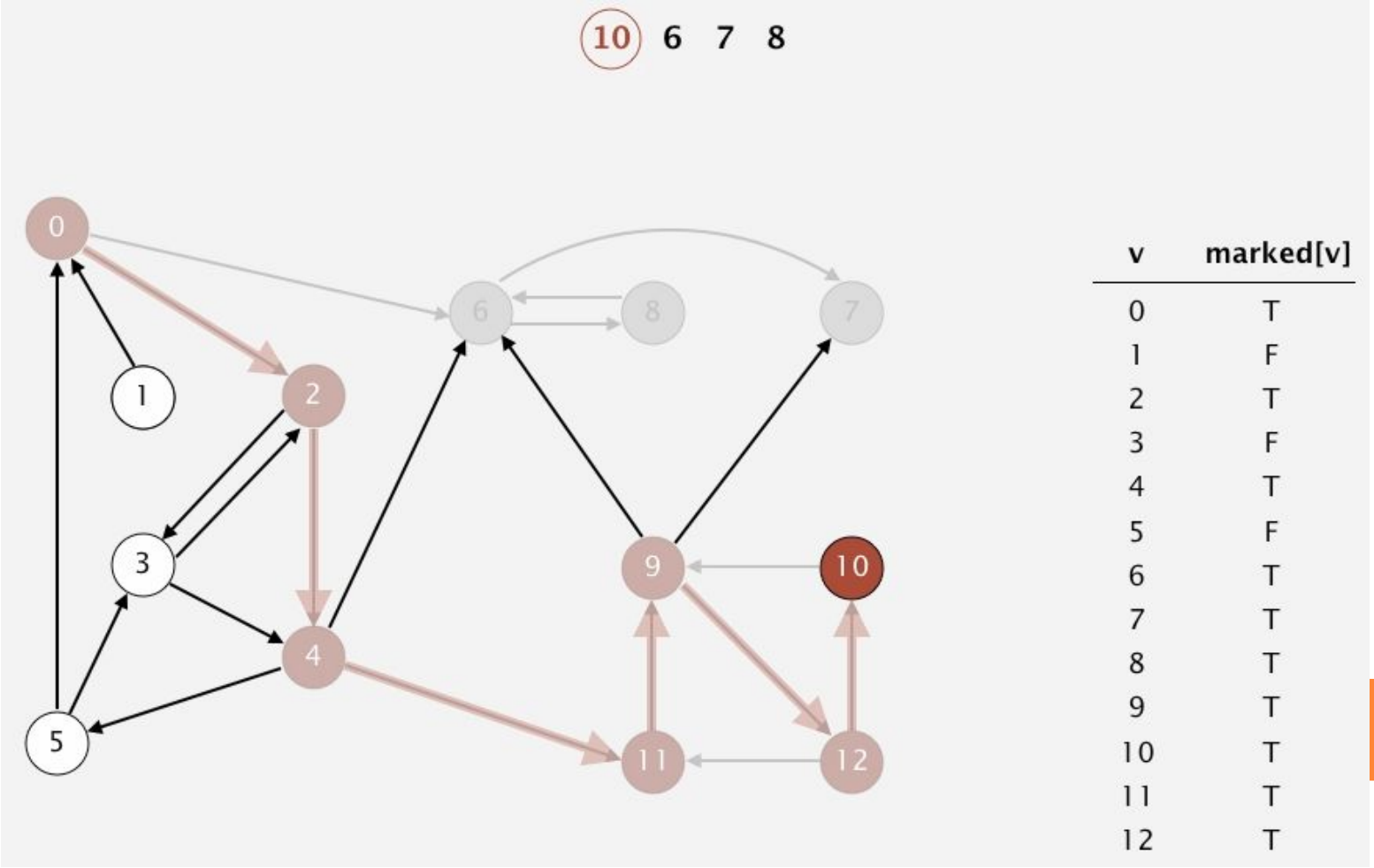
- Відвідати 12.




| v | marked[v] |
|---|---|
| 0 | T |
| 1 | F |
| 2 | T |
| 3 | F |
| 4 | T |
| 5 | F |
| 6 | T |
| 7 | T |
| 8 | T |
| 9 | T |
| 10 | F |
| 11 | T |
| 12 | T |

# Алгоритм Косараджу

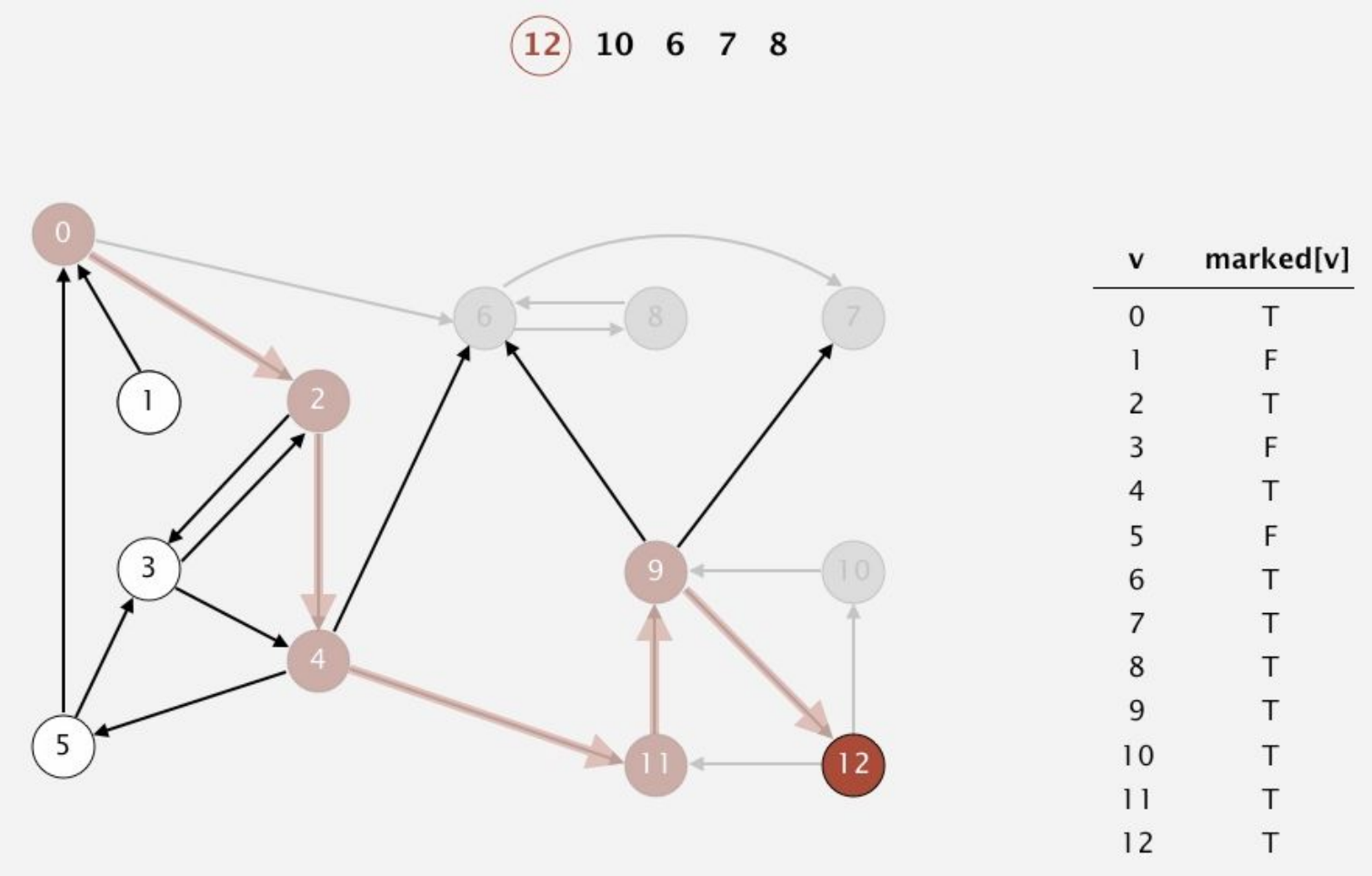
- Відвідати 10. І повернути 10.




| v | marked[v] |
|---|---|
| 0 | T |
| 1 | F |
| 2 | T |
| 3 | F |
| 4 | T |
| 5 | F |
| 6 | T |
| 7 | T |
| 8 | T |
| 9 | T |
| 10 | T |
| 11 | T |
| 12 | T |

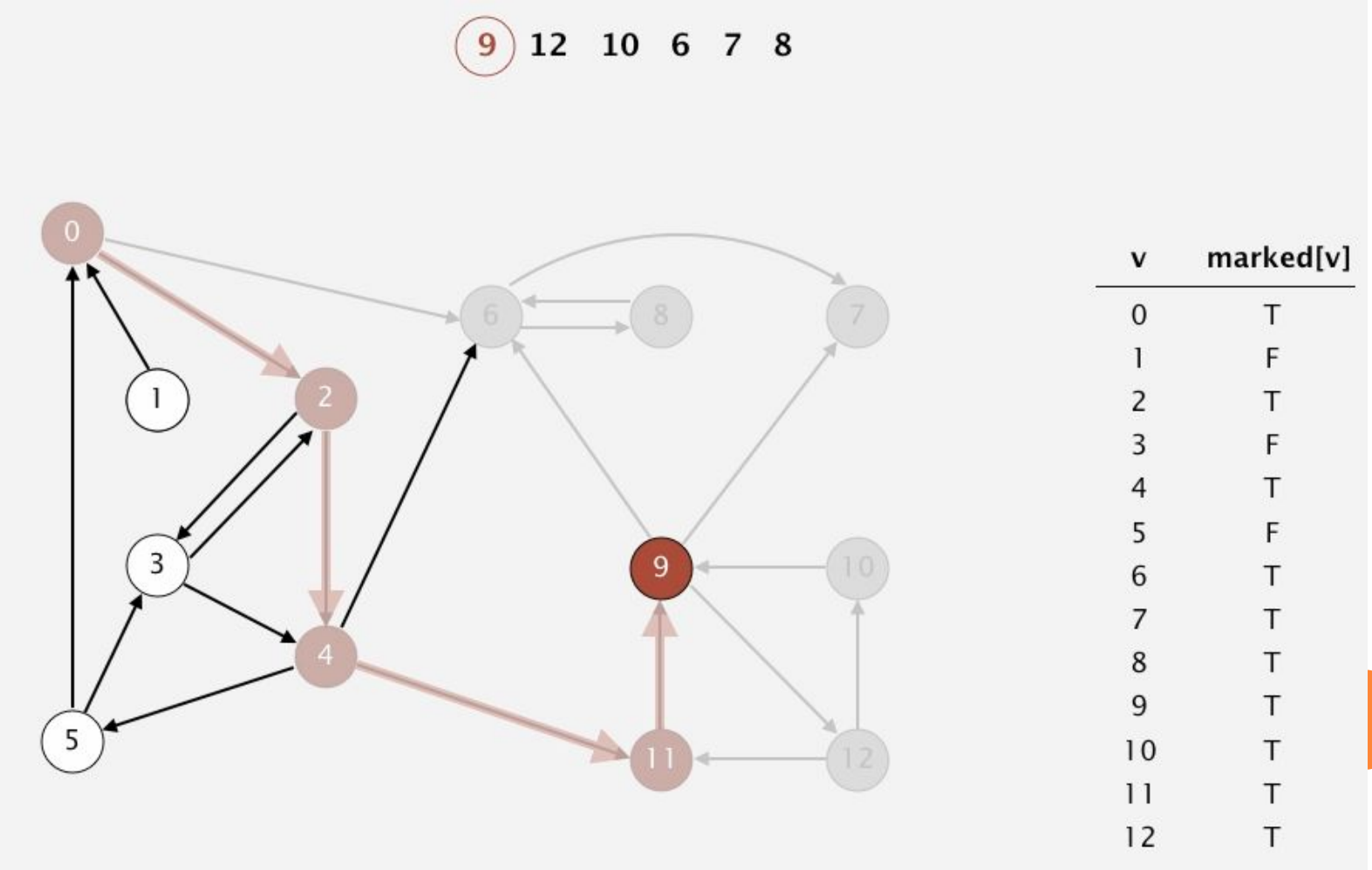
# Алгоритм Косараджу

- Повернути 12.



# Алгоритм Косараджу

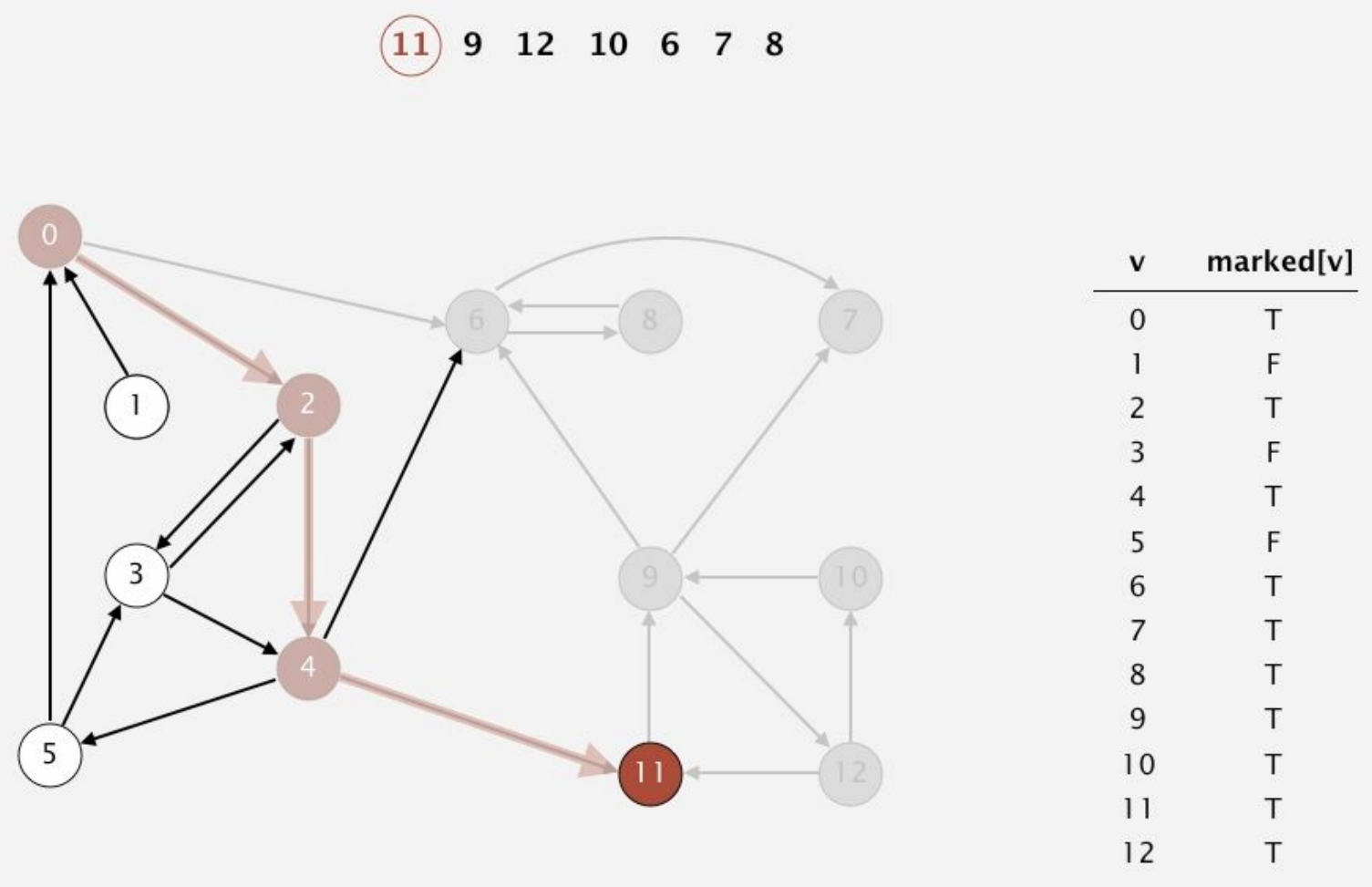
- Повернути 9.




| v | marked[v] |
|---|---|
| 0 | T |
| 1 | F |
| 2 | T |
| 3 | F |
| 4 | T |
| 5 | F |
| 6 | T |
| 7 | T |
| 8 | T |
| 9 | T |
| 10 | T |
| 11 | T |
| 12 | T |

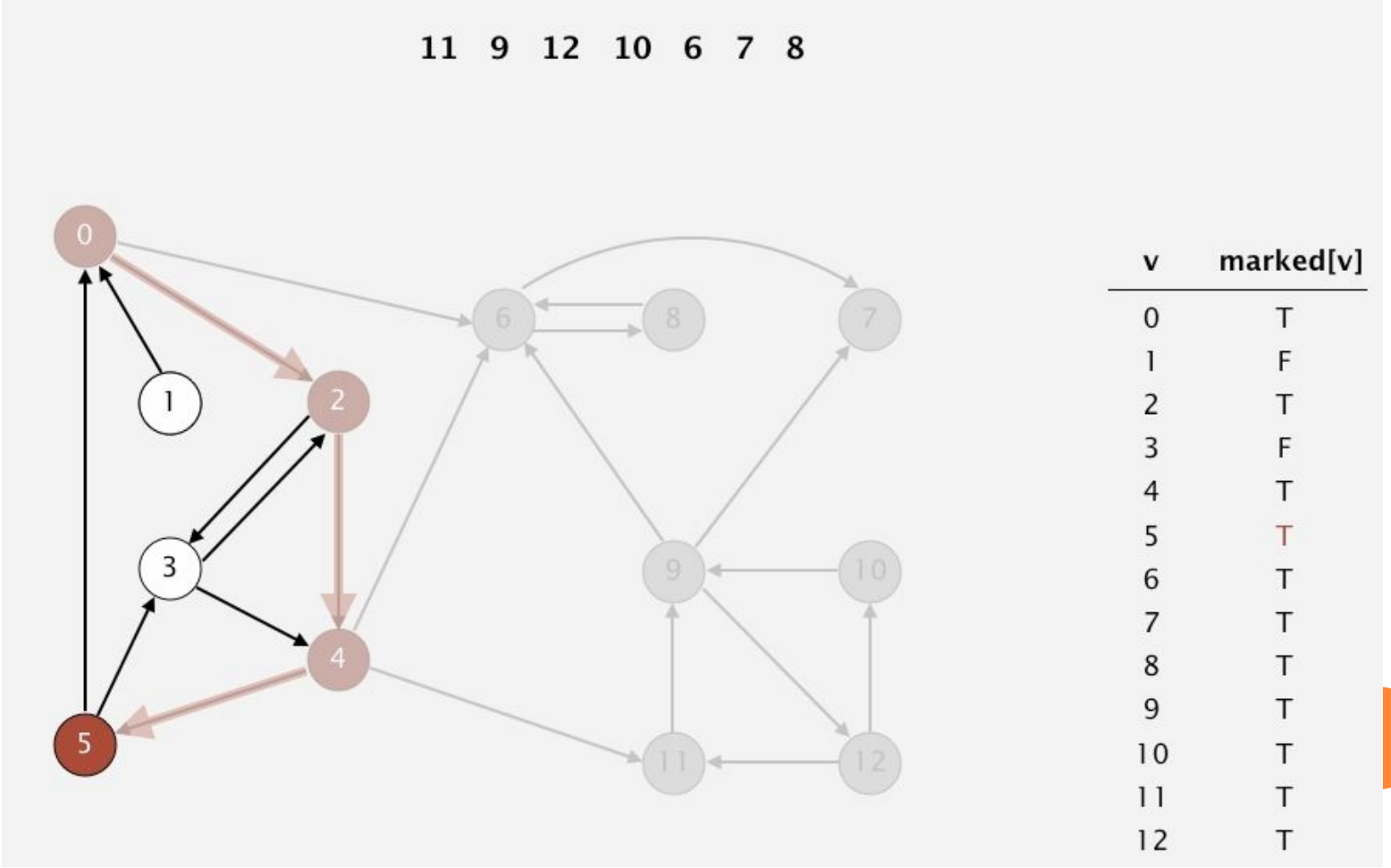
# Алгоритм Косараджу

- Повернути 11.



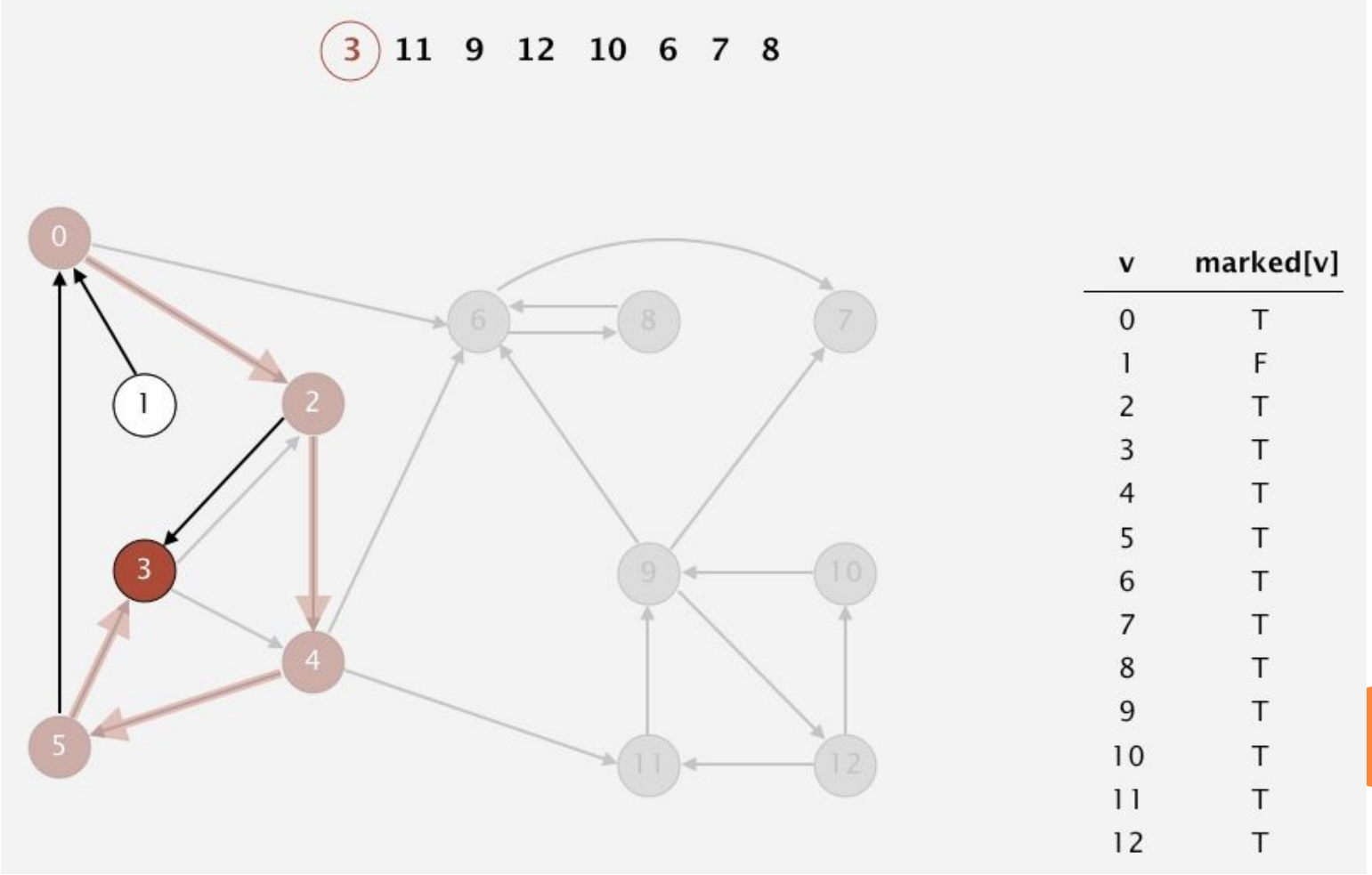
# Алгоритм Косараджу

- Відвідати 5.



# Алгоритм Косараджу

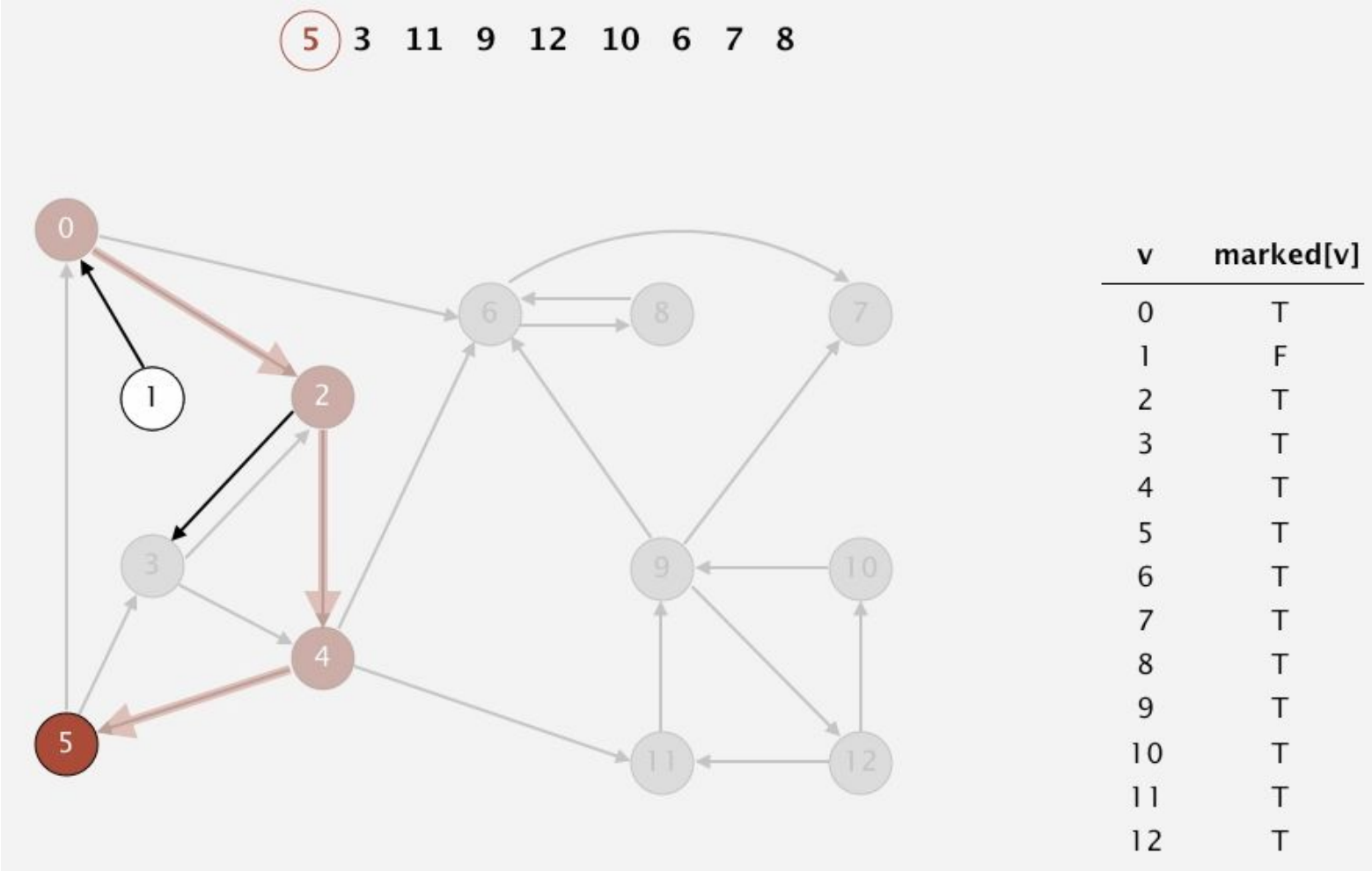
- Відвідати 3, і повернути.




| v | marked[v] |
|---|---|
| 0 | T |
| 1 | F |
| 2 | T |
| 3 | T |
| 4 | T |
| 5 | T |
| 6 | T |
| 7 | T |
| 8 | T |
| 9 | T |
| 10 | T |
| 11 | T |
| 12 | T |

# Алгоритм Косараджу

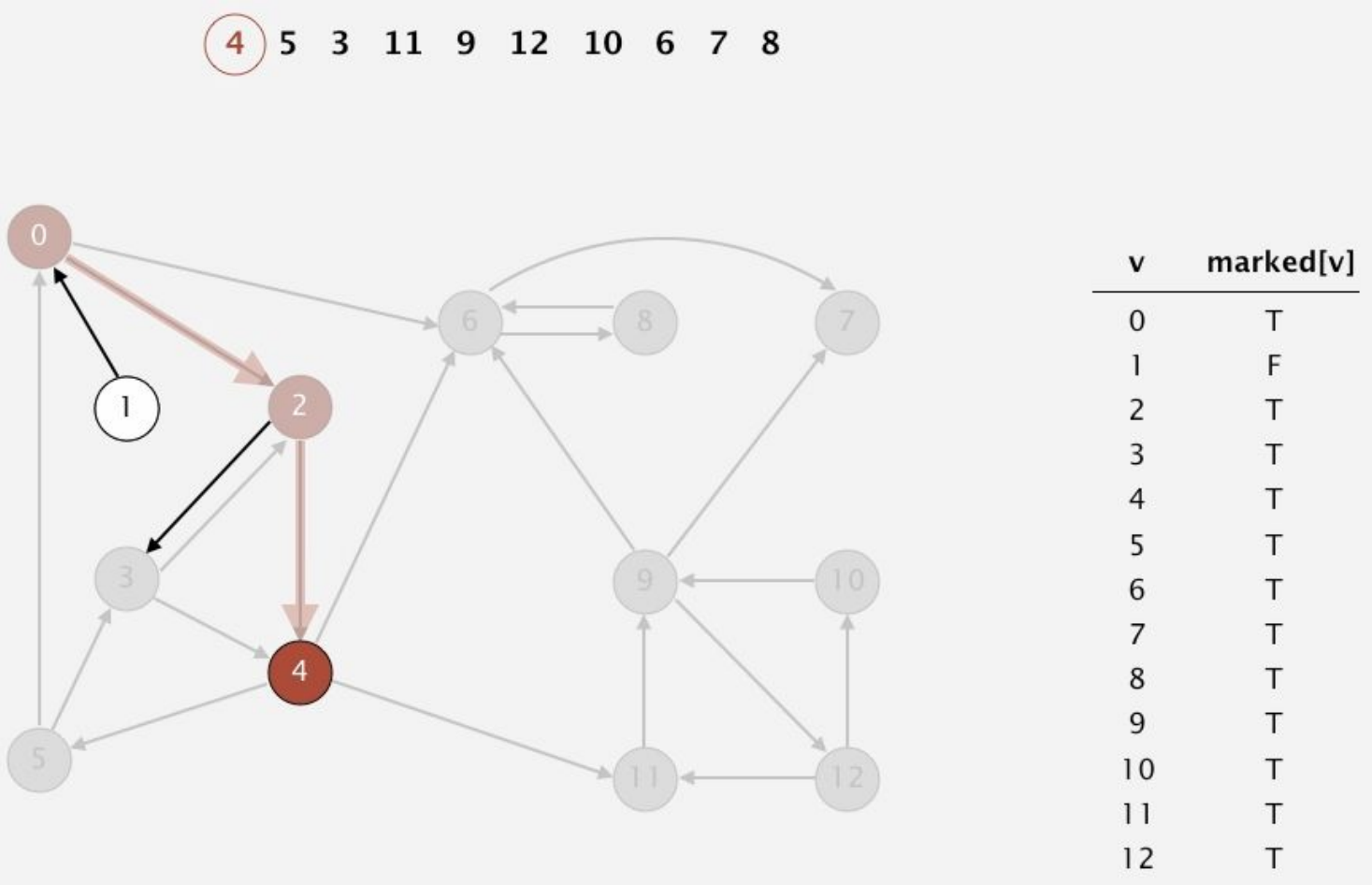
- Повернути 5.




| v | marked[v] |
|---|---|
| 0 | T |
| 1 | F |
| 2 | T |
| 3 | T |
| 4 | T |
| 5 | T |
| 6 | T |
| 7 | T |
| 8 | T |
| 9 | T |
| 10 | T |
| 11 | T |
| 12 | T |

# Алгоритм Косараджу

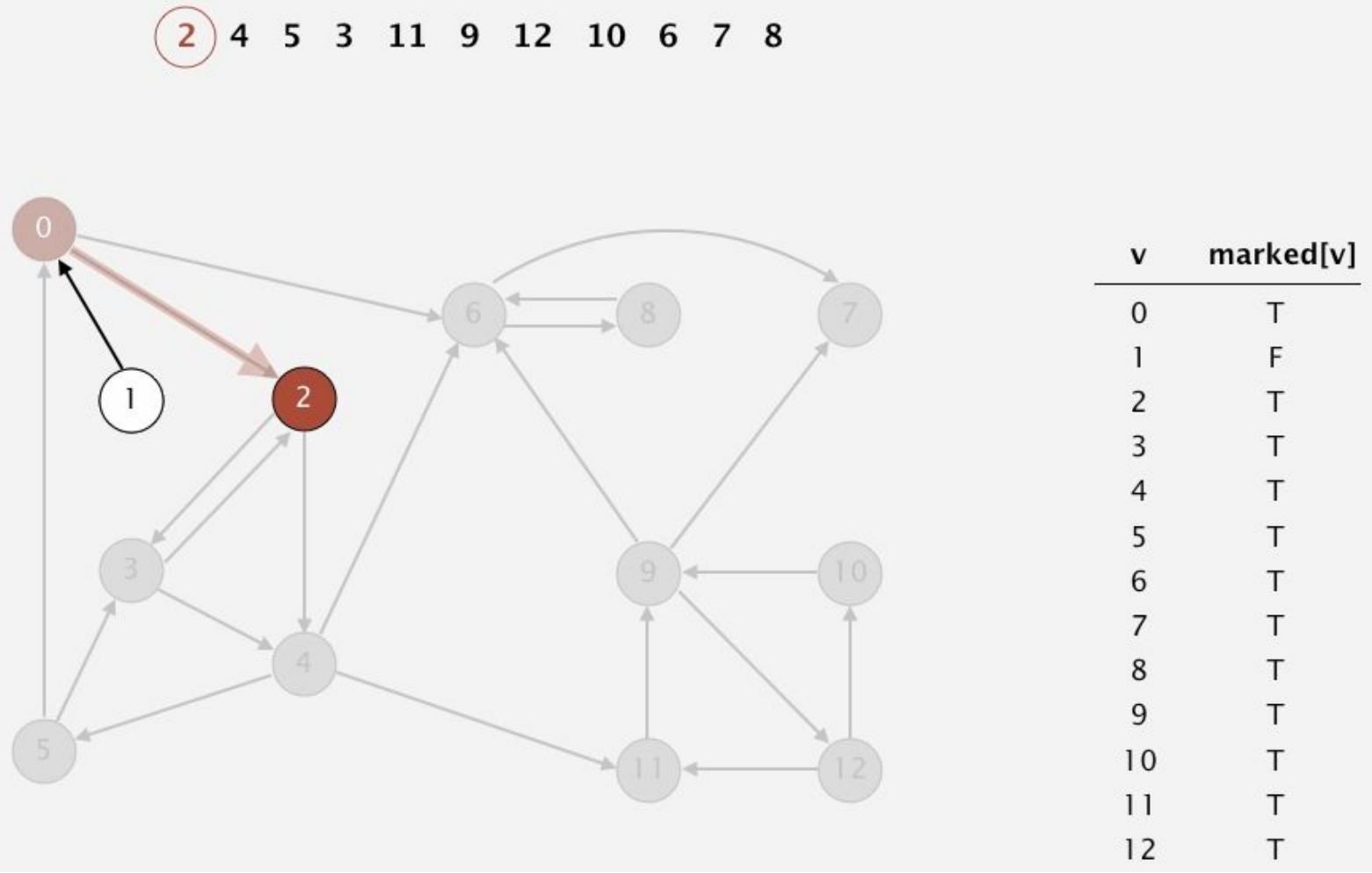
- Повернути 4.




| v | marked[v] |
|---|---|
| 0 | T |
| 1 | F |
| 2 | T |
| 3 | T |
| 4 | T |
| 5 | T |
| 6 | T |
| 7 | T |
| 8 | T |
| 9 | T |
| 10 | T |
| 11 | T |
| 12 | T |

# Алгоритм Косараджу

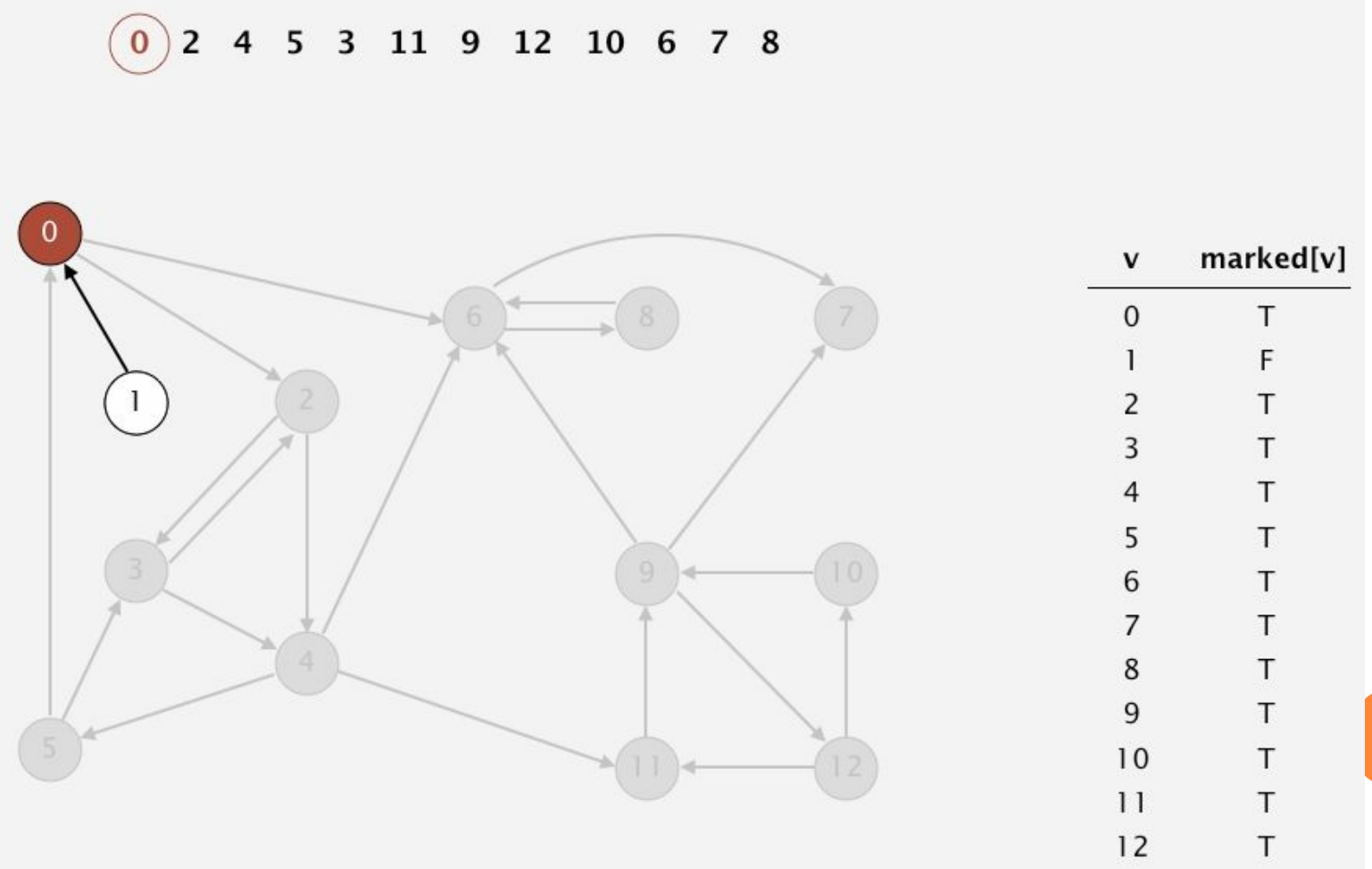
- Повернути 2.




| v | marked[v] |
|---|---|
| 0 | T |
| 1 | F |
| 2 | T |
| 3 | T |
| 4 | T |
| 5 | T |
| 6 | T |
| 7 | T |
| 8 | T |
| 9 | T |
| 10 | T |
| 11 | T |
| 12 | T |

# Алгоритм Косараджу

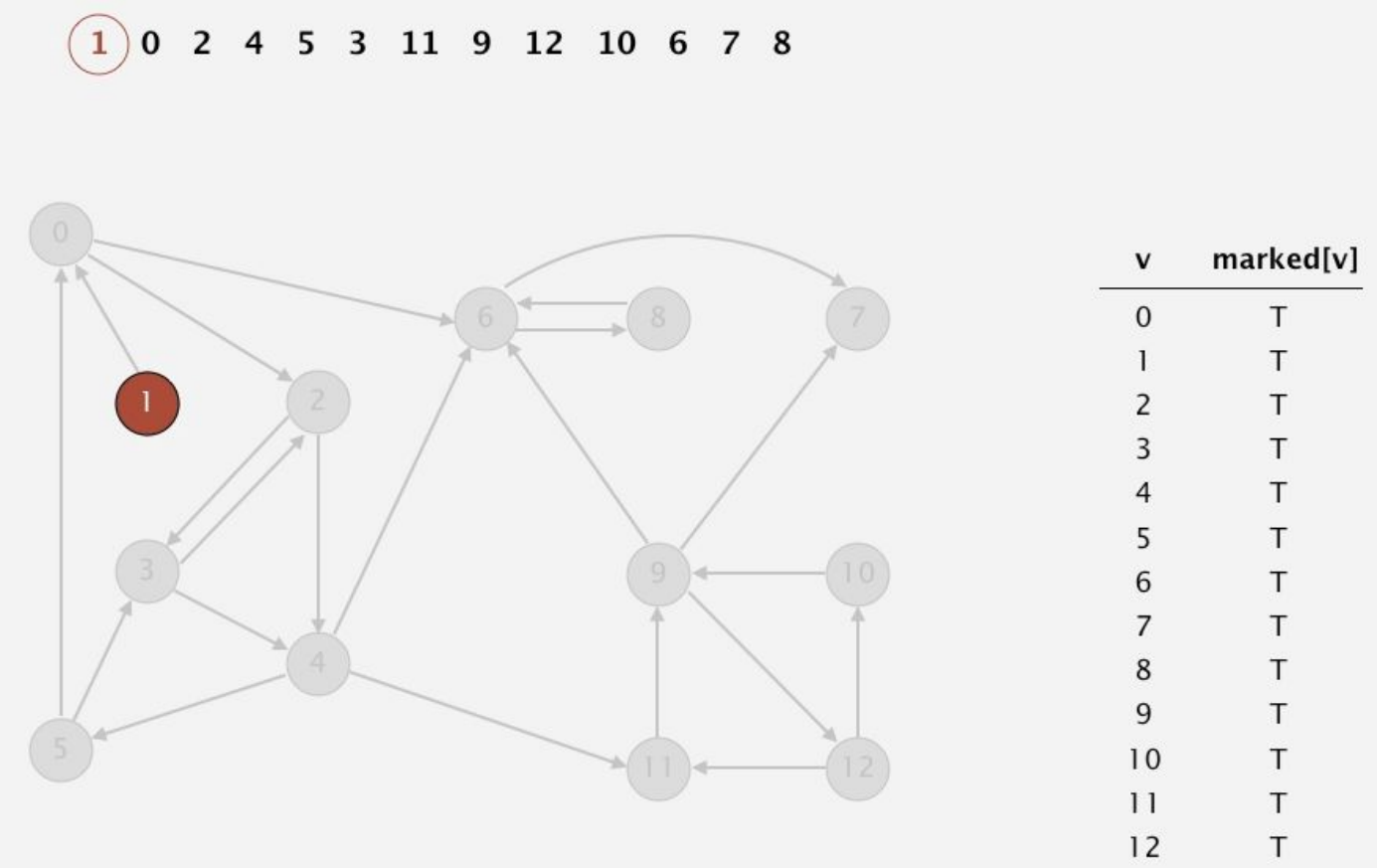
- Повернути 0.




| v | marked[v] |
|---|---|
| 0 | T |
| 1 | F |
| 2 | T |
| 3 | T |
| 4 | T |
| 5 | T |
| 6 | T |
| 7 | T |
| 8 | T |
| 9 | T |
| 10 | T |
| 11 | T |
| 12 | T |

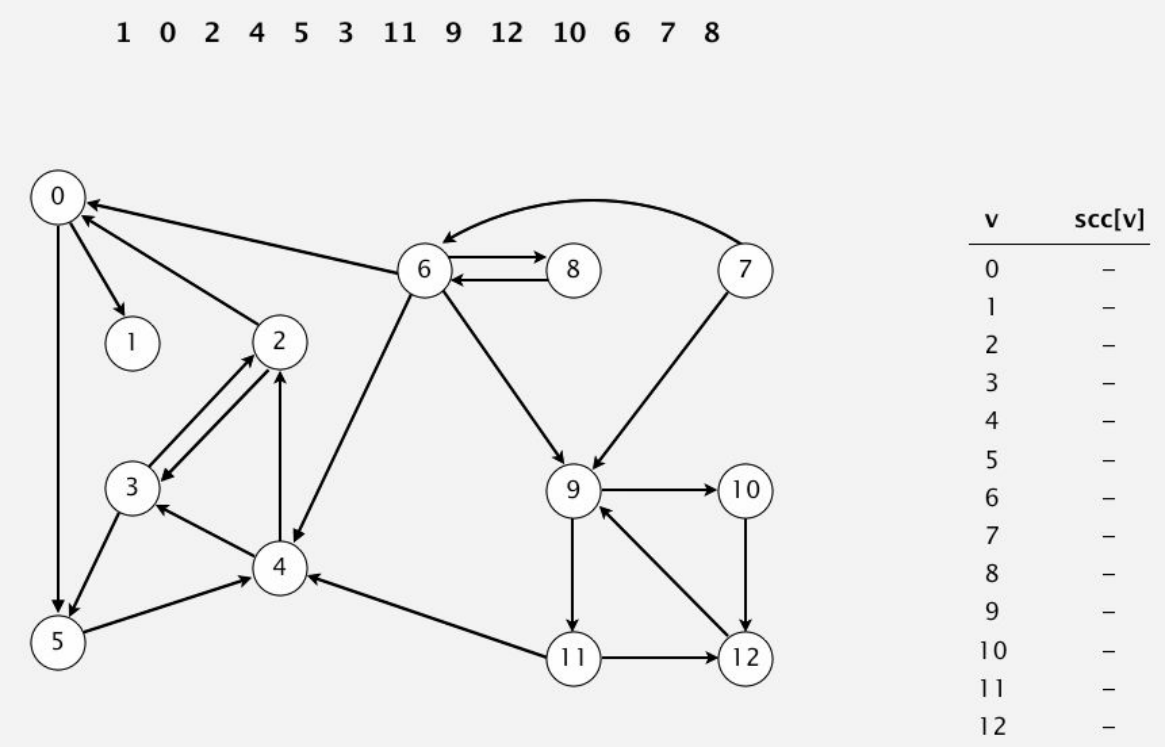
# Алгоритм Косараджу

- Єдиний вузол не відвіданий 1.
- Відвідати 1 і повернути.



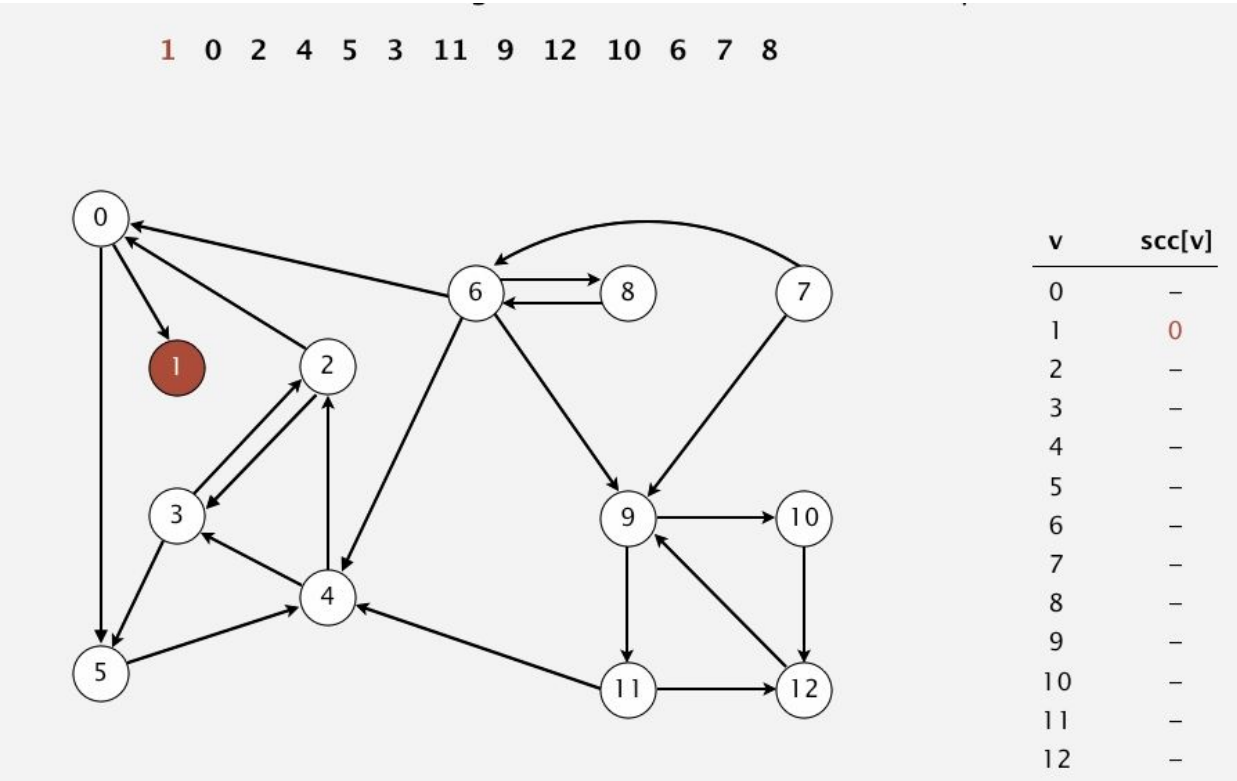
# Алгоритм Косараджу

- Ми отримали топологічну чергу.
- Перейдемо до етапу 2.
- Маємо чергу 1,0,2,4,5,3,11,9,12,10,6,7,8
- Повертаємося назад до вихідного орграфу.



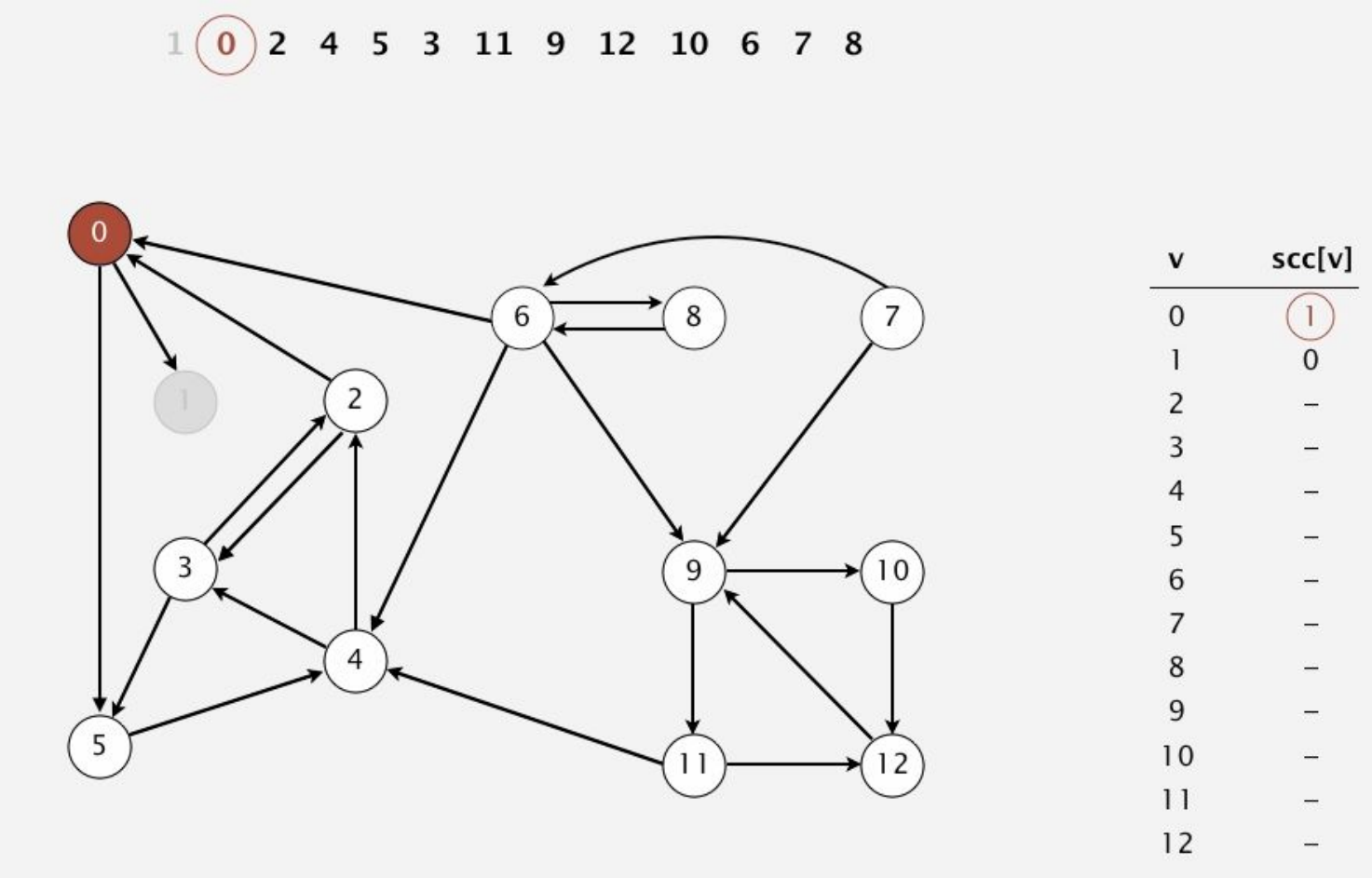
# Алгоритм Косараджу

- Тепер починаємо DFS згідно черги.
- Відвідали 1. З 1 немає більше виходів. Значить цей елемент лежить в СЗК 0.



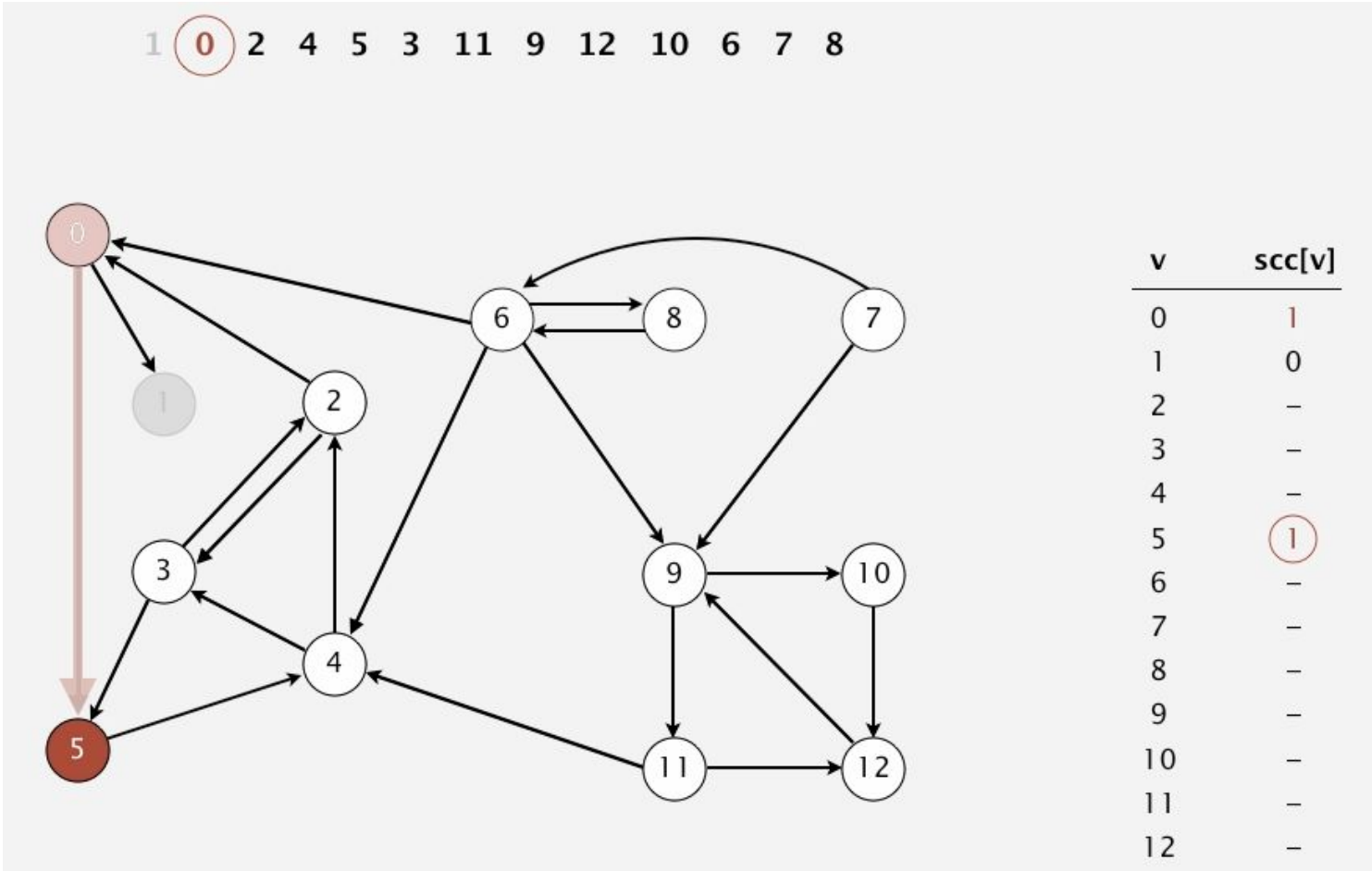
# Алгоритм Косараджу

- Відвідати 0.



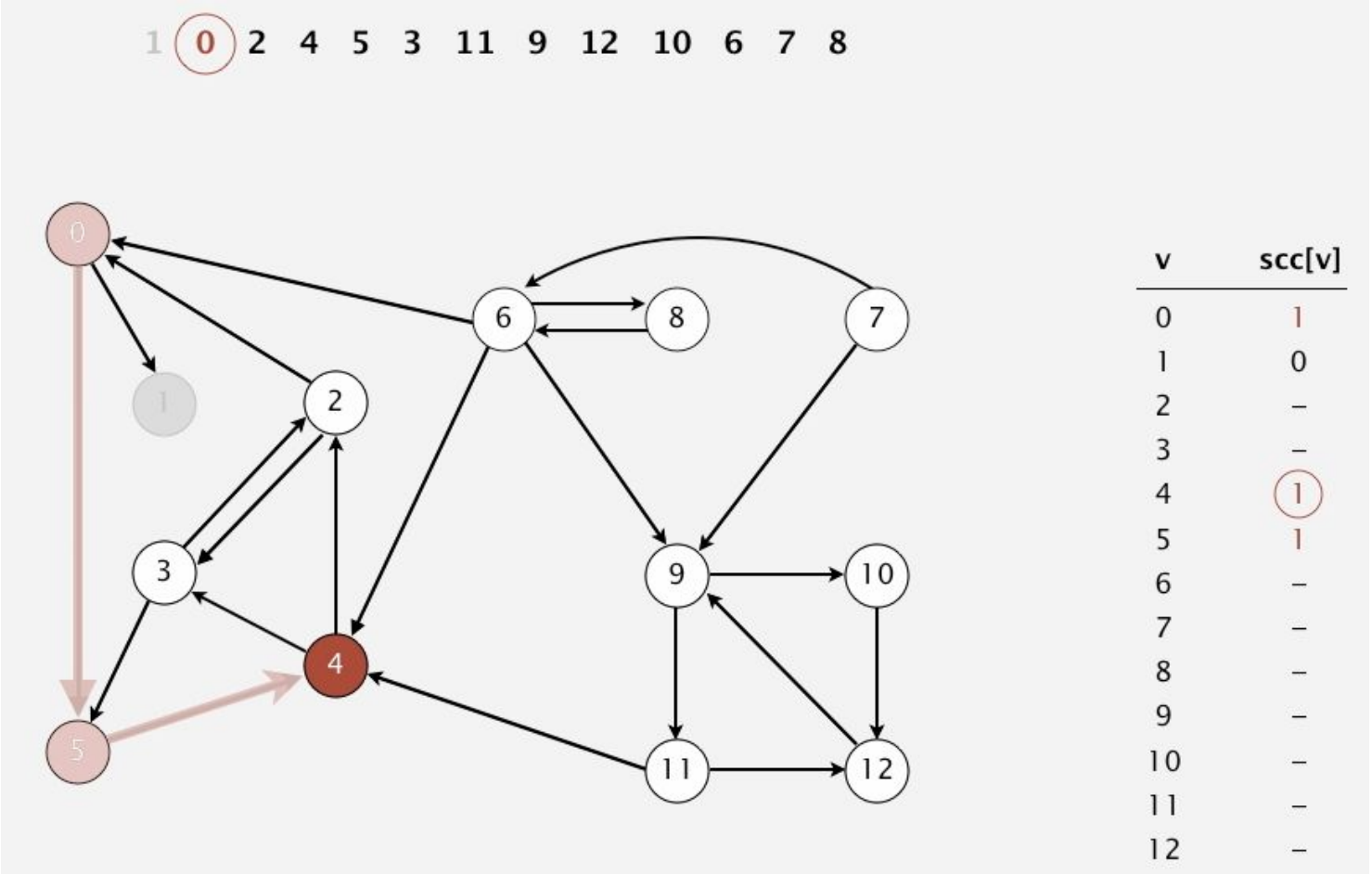
# Алгоритм Косараджу

- Відвідати 5.



# Алгоритм Косараджу

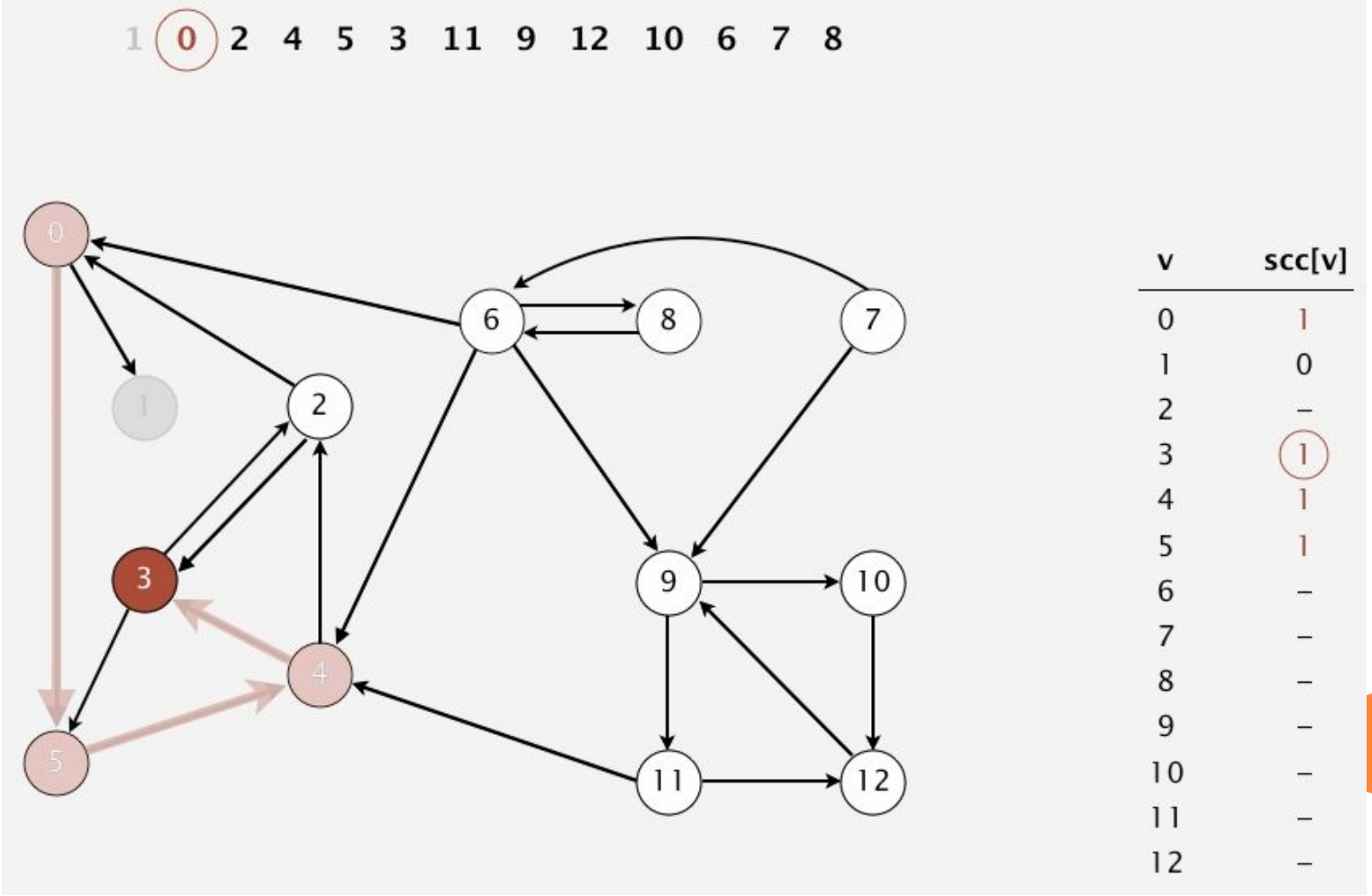
- Відвідати 4.



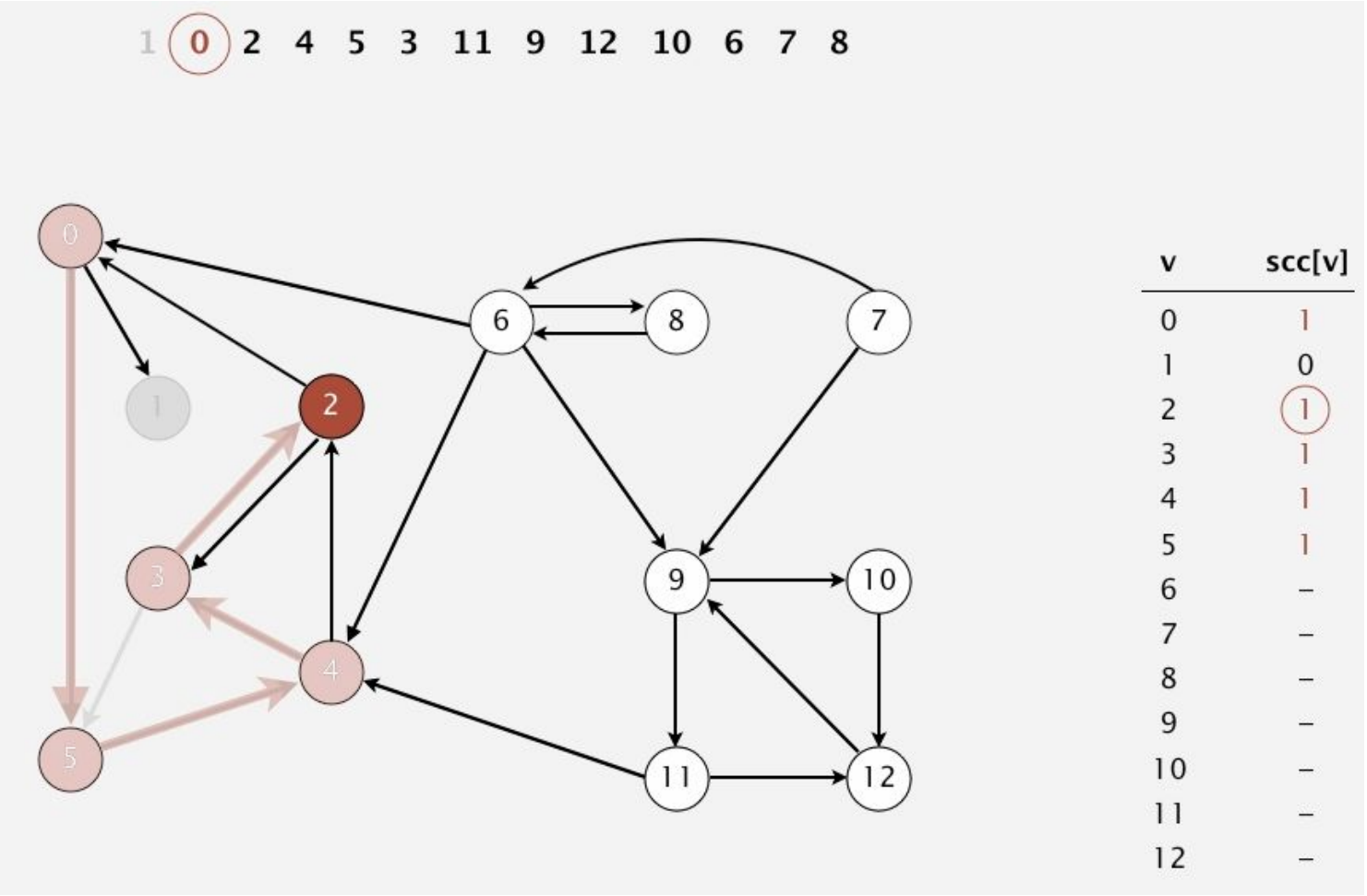

| v | scc[v] |
|---|---|
| 0 | 1 |
| 1 | 0 |
| 2 | – |
| 3 | – |
| 4 | 1 |
| 5 | 1 |
| 6 | – |
| 7 | – |
| 8 | – |
| 9 | – |
| 10 | – |
| 11 | – |
| 12 | – |

# Алгоритм Косараджу

- Відвідати 3.

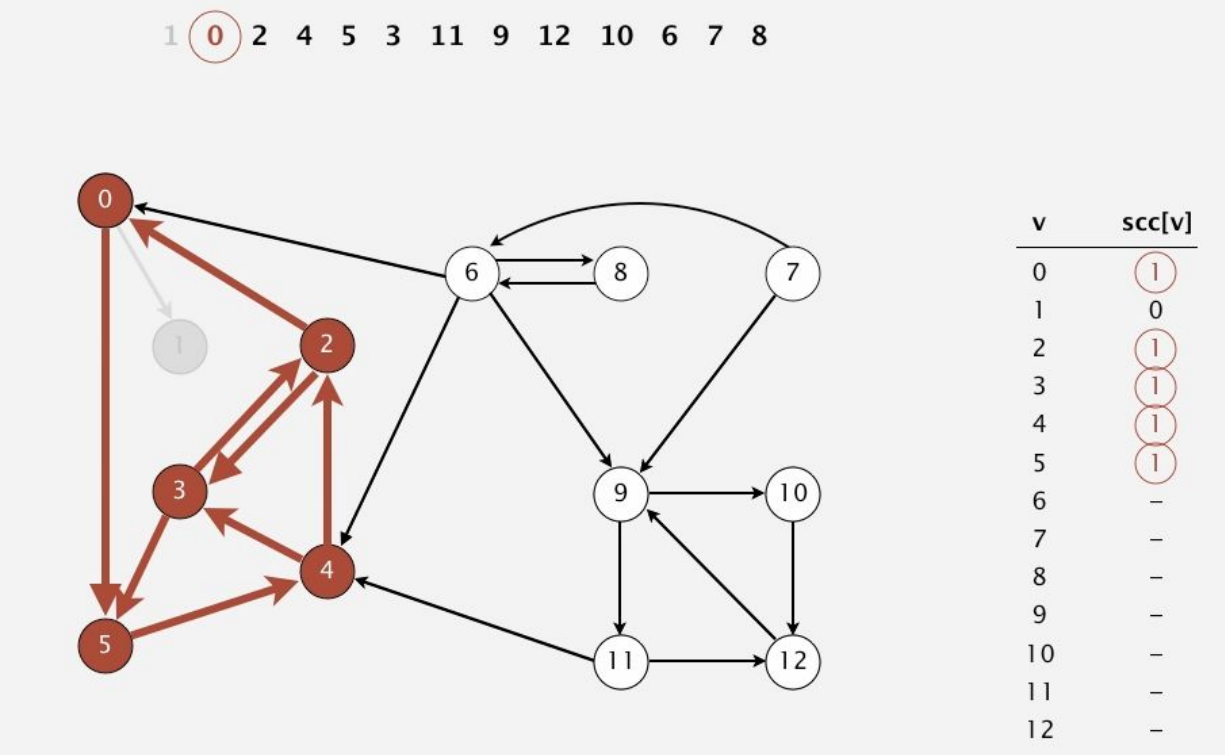


# Алгоритм Косараджу

- Відвідати 5, відвідано, відвідати 2.
- 

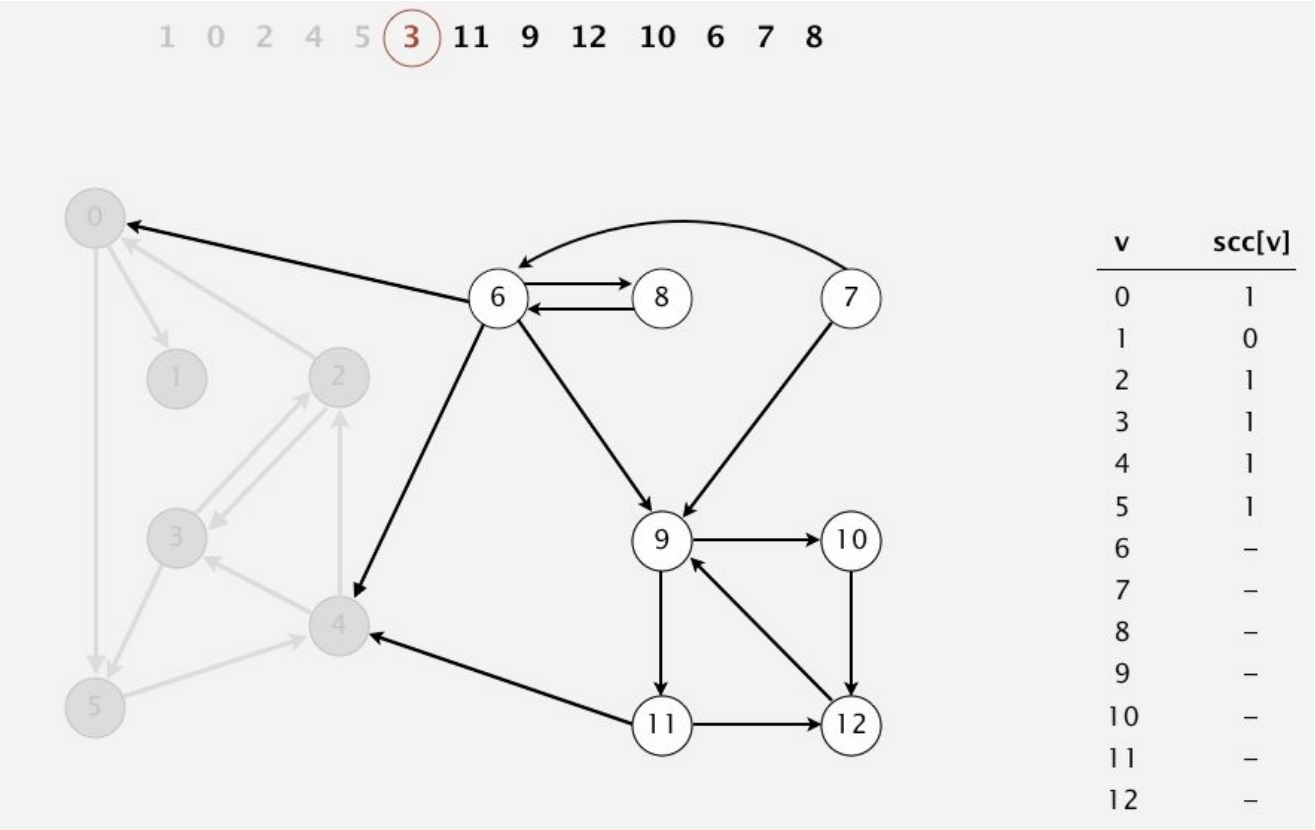
# Алгоритм Косараджу

- Відвідати 0, відвідано.
- Завершуємо крок рекурсії (повертаємося на початок).



# Алгоритм Косараджу

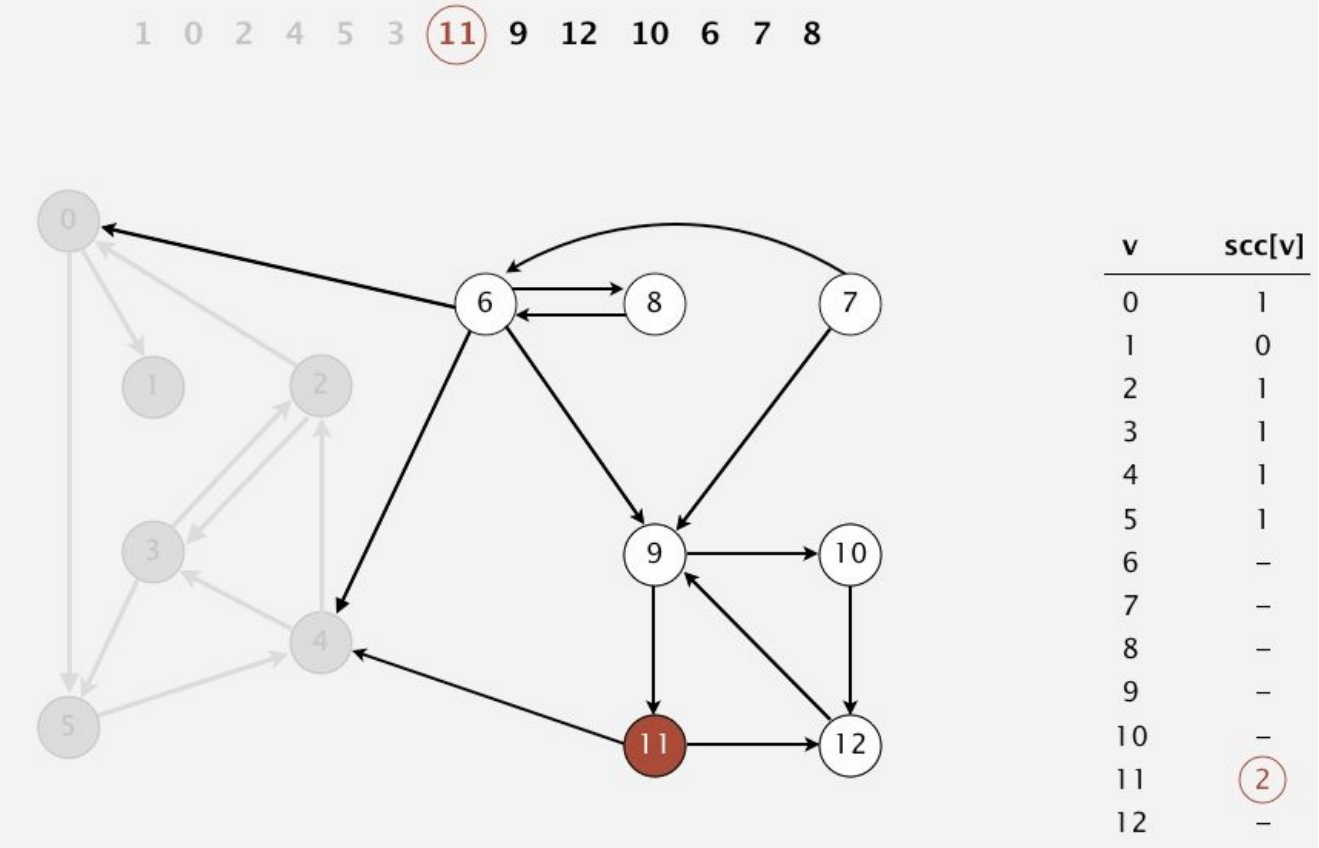
- Дістали 2 з черги. Вже відвідано, тому нічого не робимо.
- Аналогічно 4,5,3.




| v | scc[v] |
|---|---|
| 0 | 1 |
| 1 | 0 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | – |
| 7 | – |
| 8 | – |
| 9 | – |
| 10 | – |
| 11 | – |
| 12 | – |

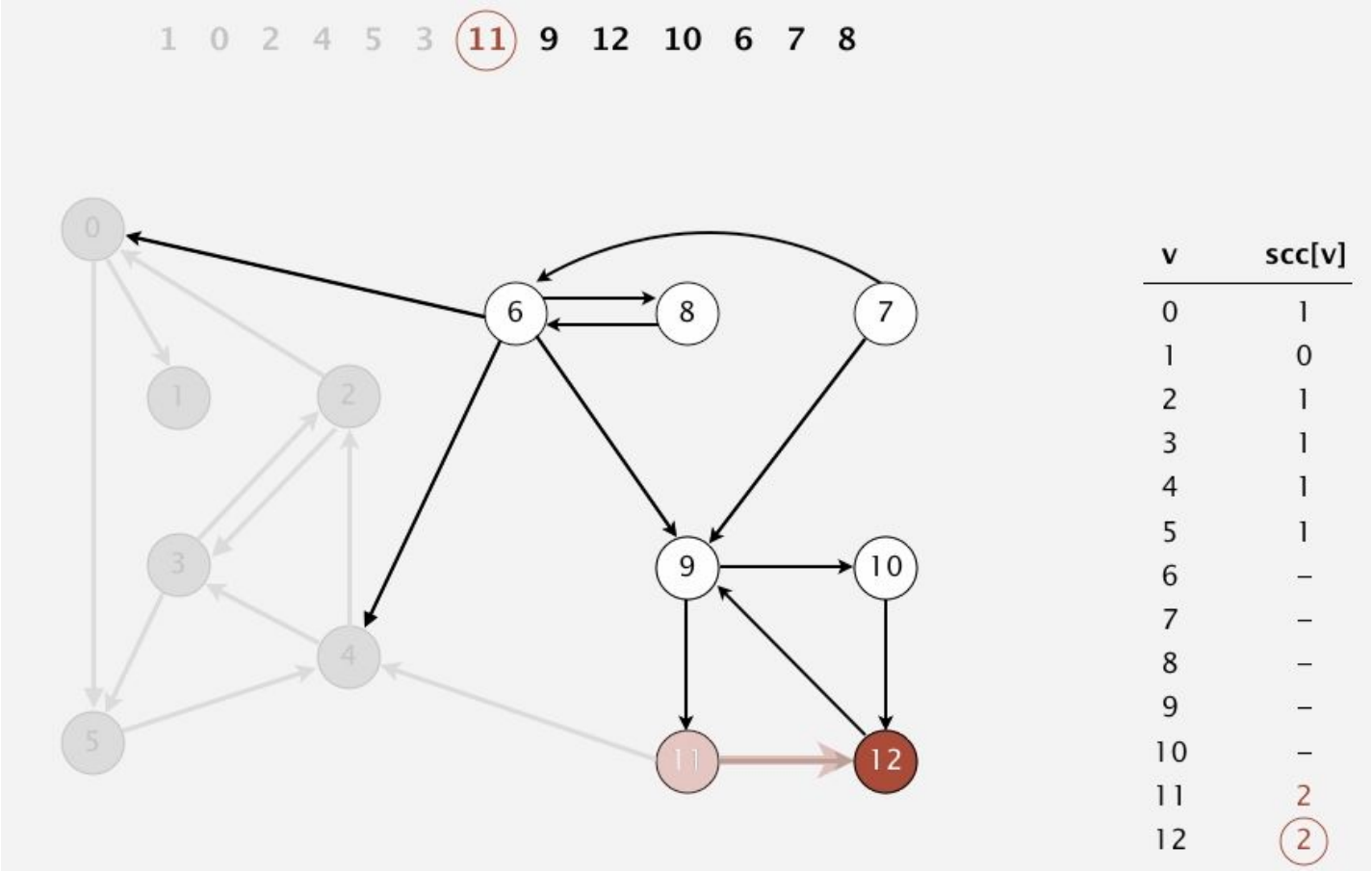
# Алгоритм Косараджу

- Забрали з черги 11.
- Відвідати 11.



# Алгоритм Косараджу

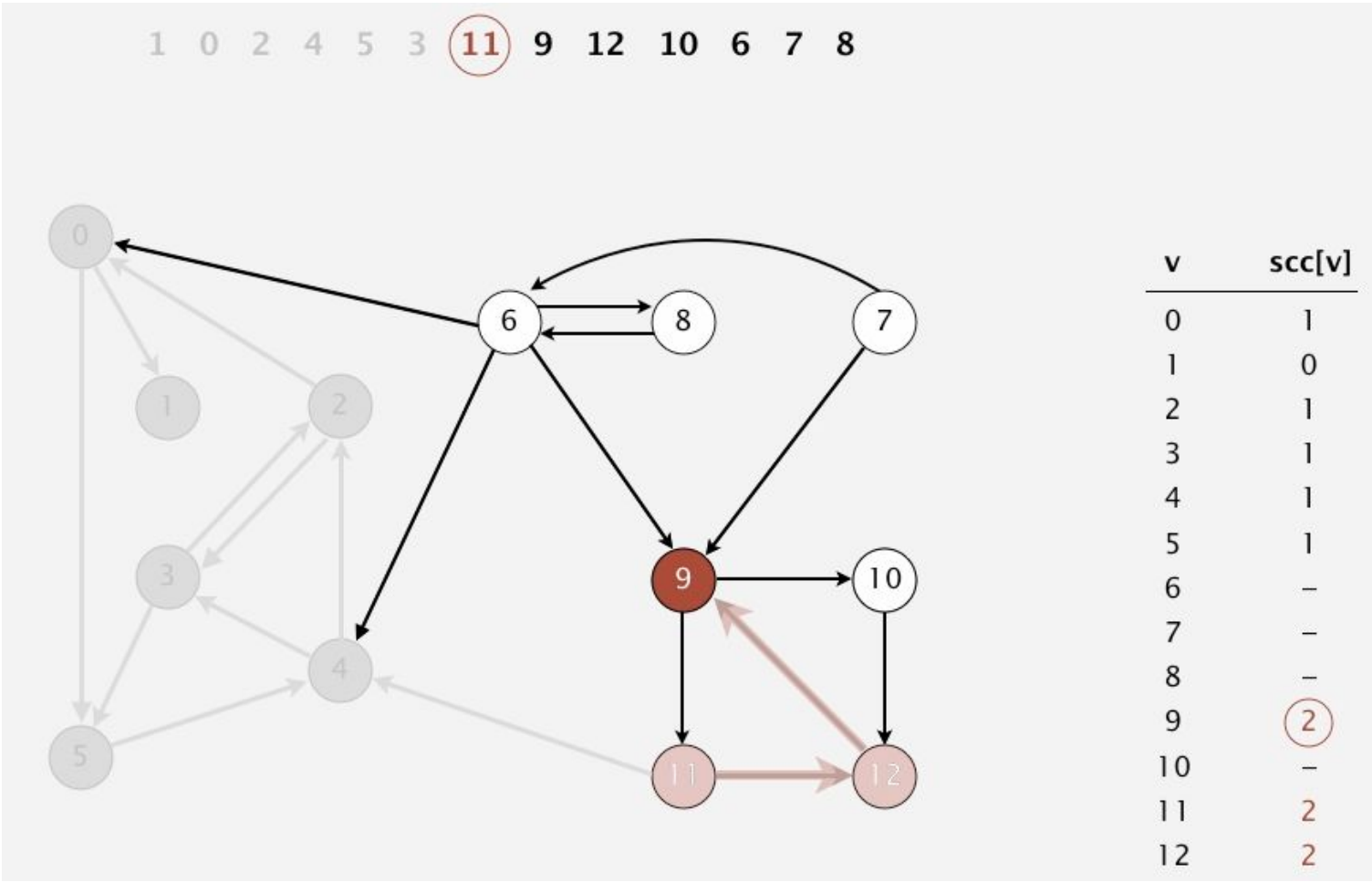
- Відвідати 12.




| v | scc[v] |
|---|---|
| 0 | 1 |
| 1 | 0 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | – |
| 7 | – |
| 8 | – |
| 9 | – |
| 10 | – |
| 11 | 2 |
| 12 | 2 |

# Алгоритм Косараджу

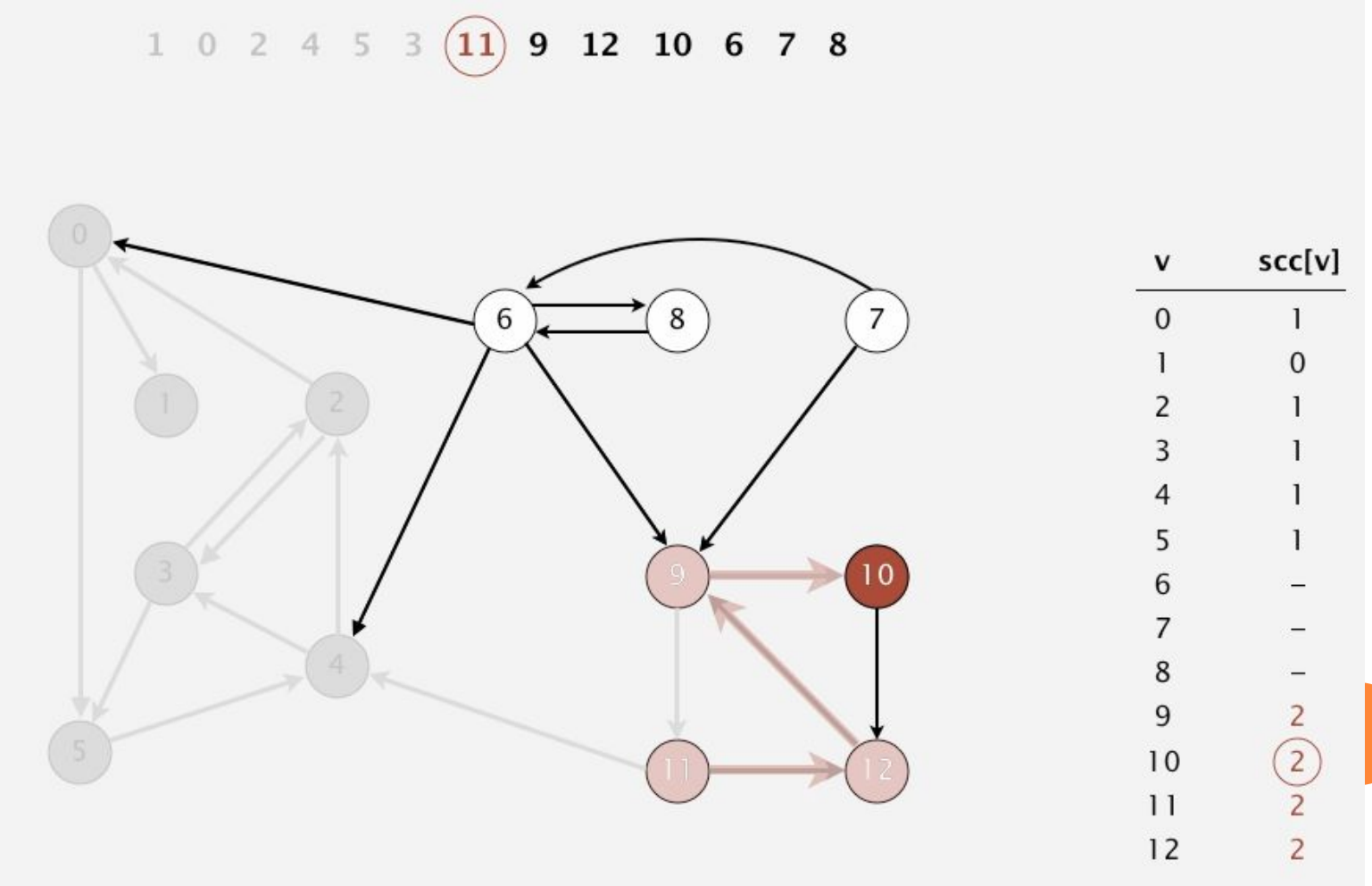
- Відвідати 9.




| v | scc[v] |
|---|---|
| 0 | 1 |
| 1 | 0 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | – |
| 7 | – |
| 8 | – |
| 9 | 2 |
| 10 | – |
| 11 | 2 |
| 12 | 2 |

# Алгоритм Косараджу

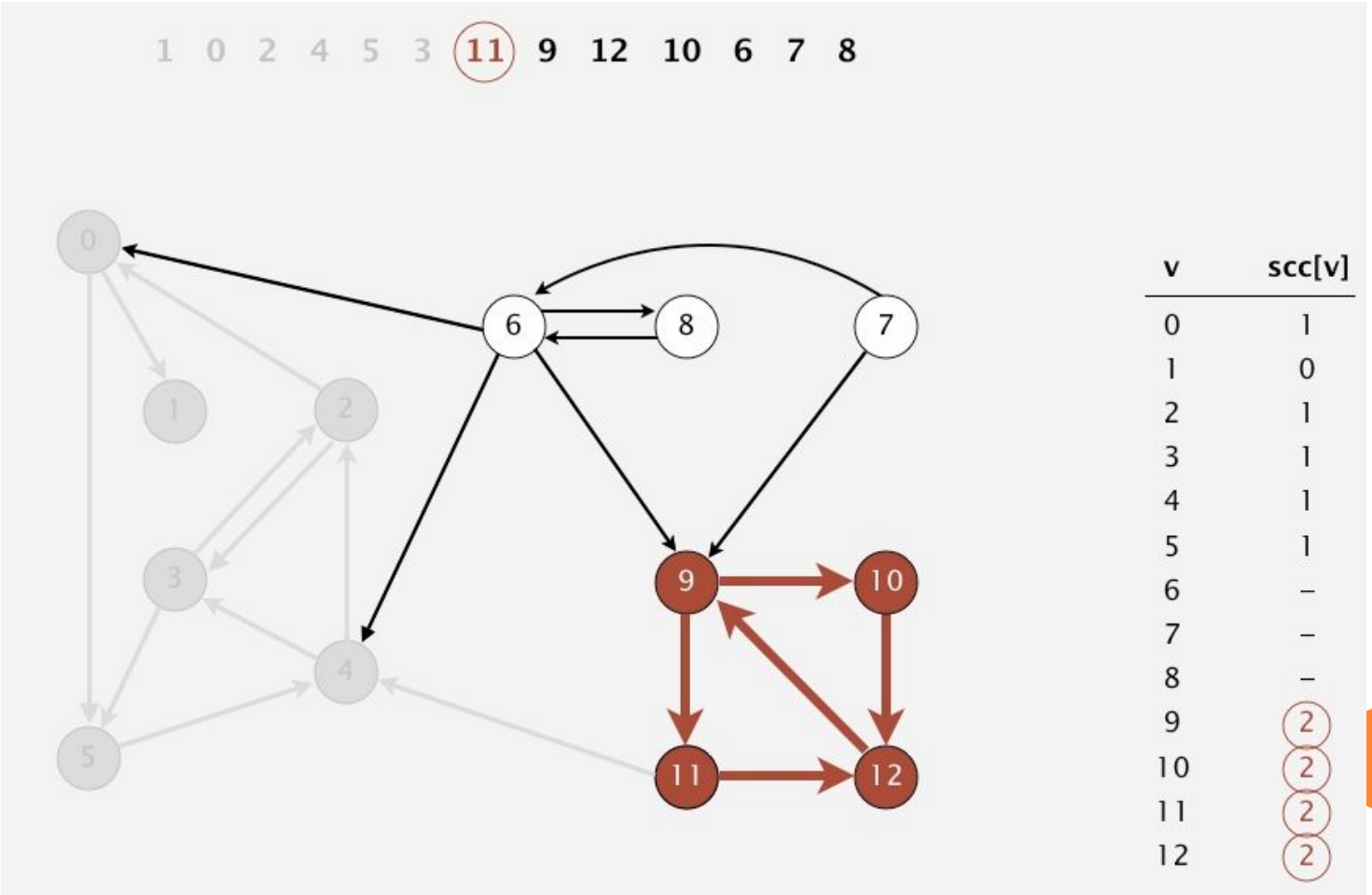
- Відвідати 10.




| v | scc[v] |
|---|---|
| 0 | 1 |
| 1 | 0 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | – |
| 7 | – |
| 8 | – |
| 9 | 2 |
| 10 | 2 |
| 11 | 2 |
| 12 | 2 |

# Алгоритм Косараджу

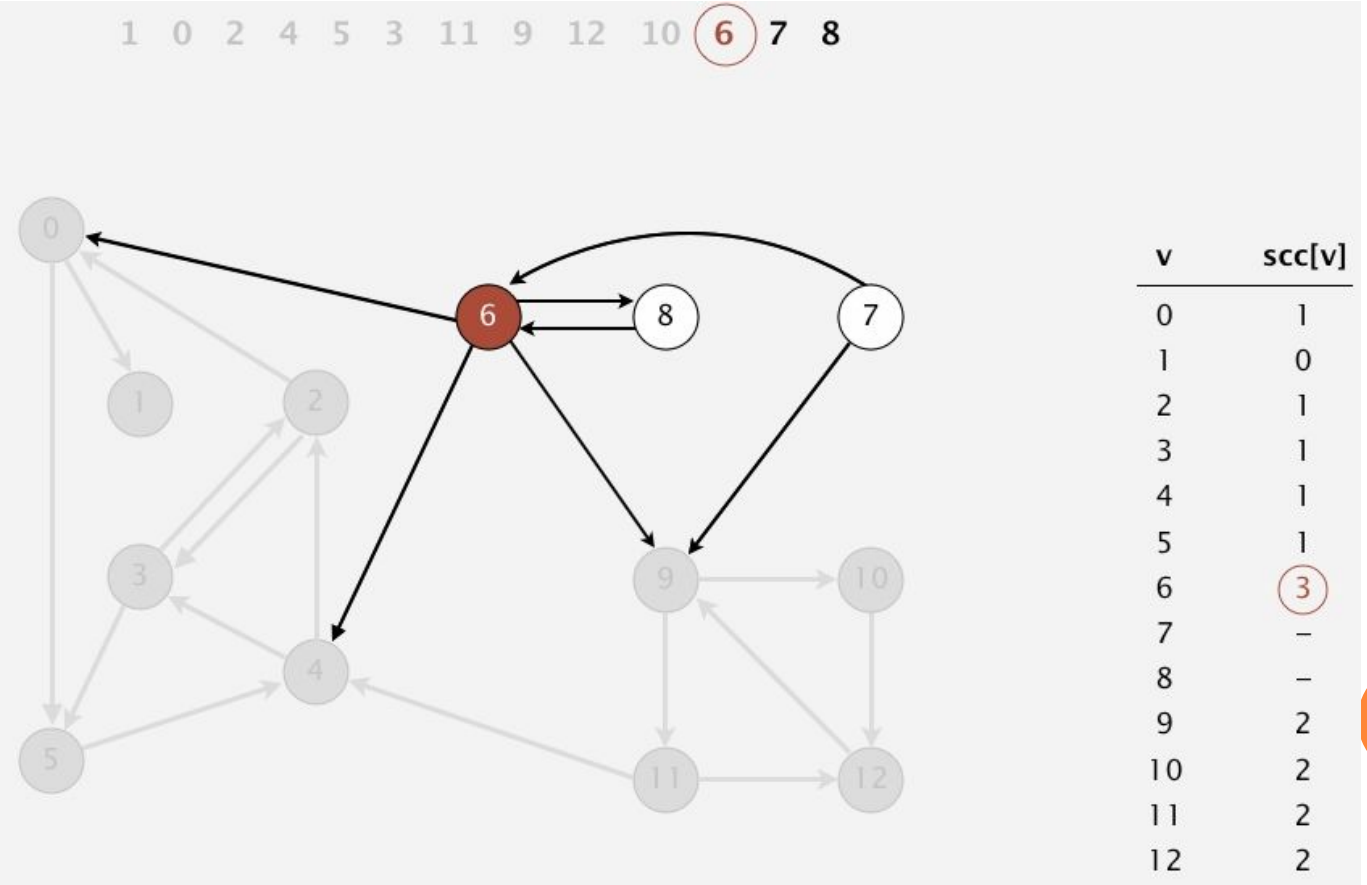
- Всі можливі вузли відвідані.




| v | scc[v] |
|---|---|
| 0 | 1 |
| 1 | 0 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | – |
| 7 | – |
| 8 | – |
| 9 | 2 |
| 10 | 2 |
| 11 | 2 |
| 12 | 2 |

# Алгоритм Косараджу

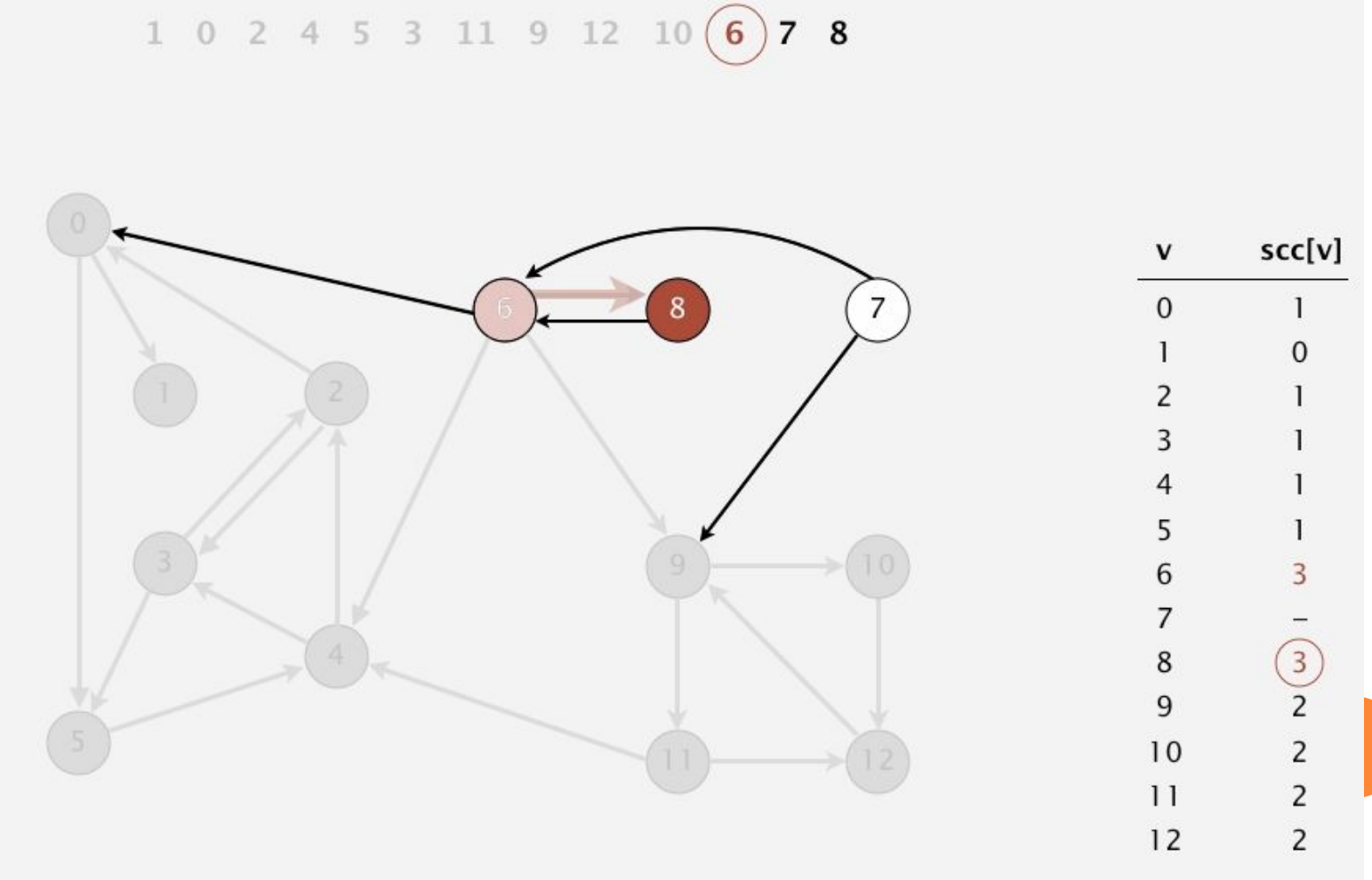
- Забираємо з черги 9,12,10, вони всі відвідані.
- Забираємо 6 і відвідуємо.




| v | scc[v] |
|---|---|
| 0 | 1 |
| 1 | 0 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | 3 |
| 7 | – |
| 8 | – |
| 9 | 2 |
| 10 | 2 |
| 11 | 2 |
| 12 | 2 |

# Алгоритм Косараджу

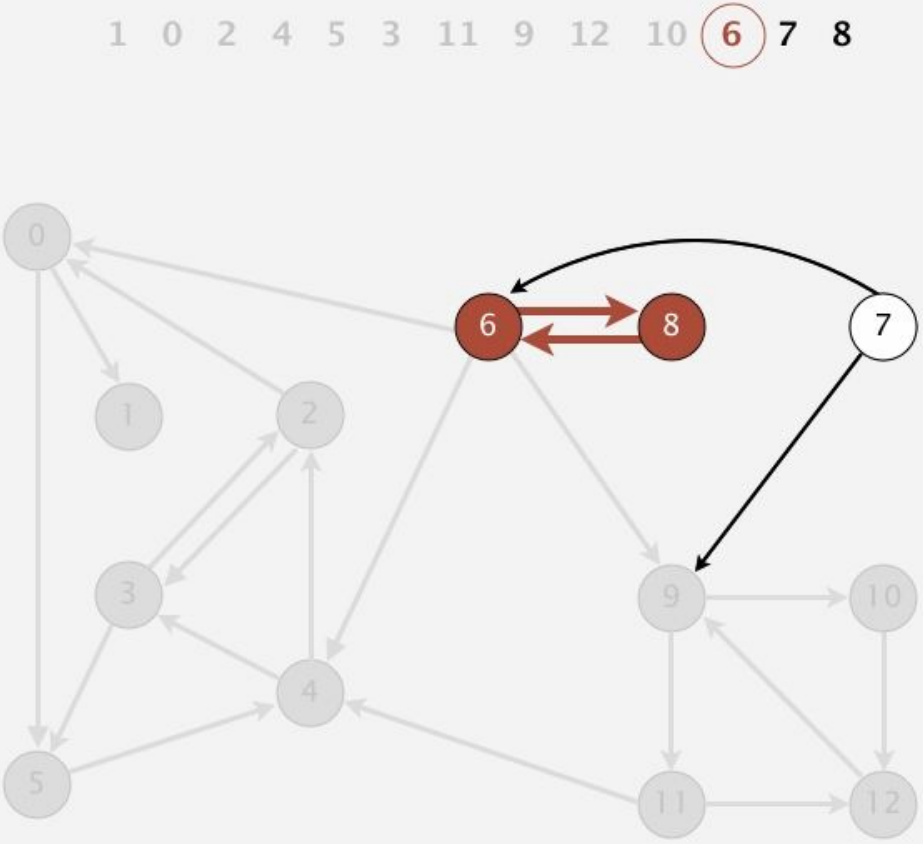
- Відвідати 8.




| v | scc[v] |
|---|---|
| 0 | 1 |
| 1 | 0 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | 3 |
| 7 | – |
| 8 | 3 |
| 9 | 2 |
| 10 | 2 |
| 11 | 2 |
| 12 | 2 |

# Алгоритм Косараджу

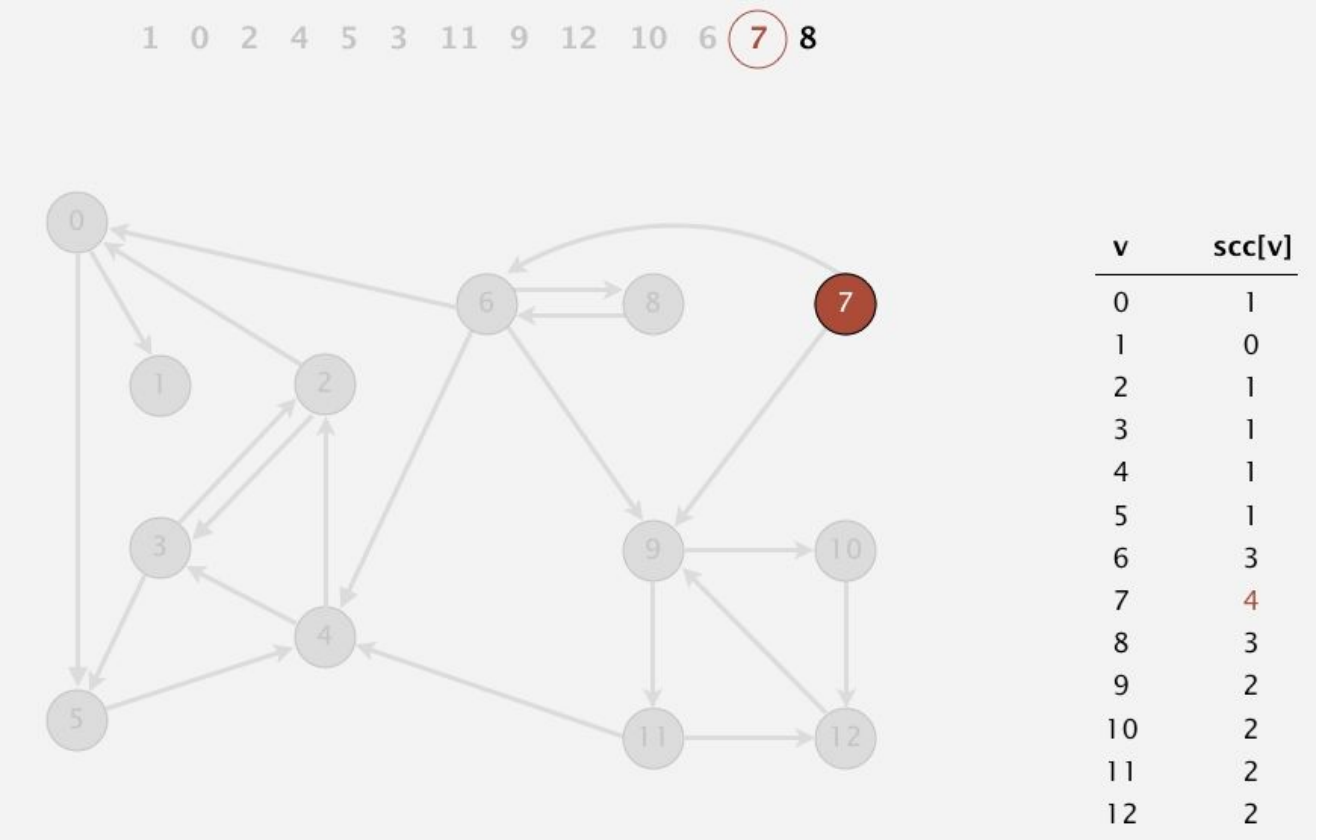
- Отримали ще один СЗК




| v | scc[v] |
|---|---|
| 0 | 1 |
| 1 | 0 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | 3 |
| 7 | – |
| 8 | 3 |
| 9 | 2 |
| 10 | 2 |
| 11 | 2 |
| 12 | 2 |

# Алгоритм Косараджу

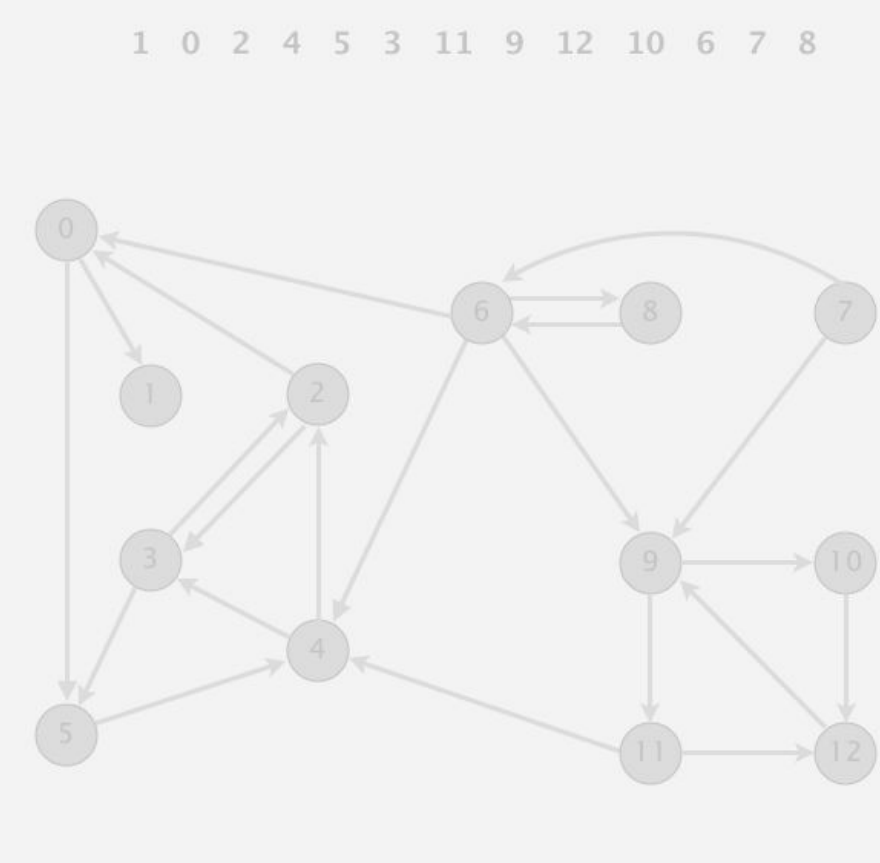
- Забираємо з черги 7 і відвідуємо.
- Цей елемент один утворює СЗК.




| v | scc[v] |
|---|---|
| 0 | 1 |
| 1 | 0 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | 3 |
| 7 | 4 |
| 8 | 3 |
| 9 | 2 |
| 10 | 2 |
| 11 | 2 |
| 12 | 2 |

# Алгоритм Косараджу

- Забираємо з черги 8. Воно відвідано, закінчили роботу.




| v | scc[v] |
|---|---|
| 0 | 1 |
| 1 | 0 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | 3 |
| 7 | 4 |
| 8 | 3 |
| 9 | 2 |
| 10 | 2 |
| 11 | 2 |
| 12 | 2 |

# Алгоритм Косараджу

- Алгоритм Косараджу обчислює СЗКти орграфа за час пропорційний E+V.
  - Вузьке місце – запуск DFS двічі і обрахунок транспонованого графу.
  - В принципі можна усунути.

  - Реалізація дуже проста.
  - Необхідно змінити пару стрічок.

# Алгоритм Косараджу

```
public class CC{
    private boolean marked[];
    private int[] id;
    private int count;
    public CC(Graph G){
        marked = new boolean[G.V()];
        id = new int[G.V()];
        for (int v = 0; v < G.V(); v++){
            if (!marked[v]){
                dfs(G, v);
                count++;
            }
        }
    }
    private void dfs(Graph G, int v){
        marked[v] = true;
        id[v] = count;
        for (int w : G.adj(v))
            if (!marked[w])
                dfs(G, w);
    }
    public boolean connected(int v, int w){ return id[v] == id[w]; }
}
```

# Алгоритм Косараджу

```
public class KosarajuSharirSCC{
    private boolean marked[];
    private int[] id;
    private int count;
    public KosarajuSharirSCC(Digraph G){
        marked = new boolean[G.V()];
        id = new int[G.V()];
        DepthFirstOrder dfs = new DepthFirstOrder(G.reverse());
        for (int v : dfs.reversePost()){
            if (!marked[v]){
                dfs(G, v);
                count++;
            }
        }
    }
    private void dfs(Digraph G, int v){
        marked[v] = true;
        id[v] = count;
        for (int w : G.adj(v))
            if (!marked[w])
                dfs(G, w);
        }
    public boolean stronglyConnected(int v, int w){ return id[v] == id[w]; }
}
```

- Дякую за увагу.