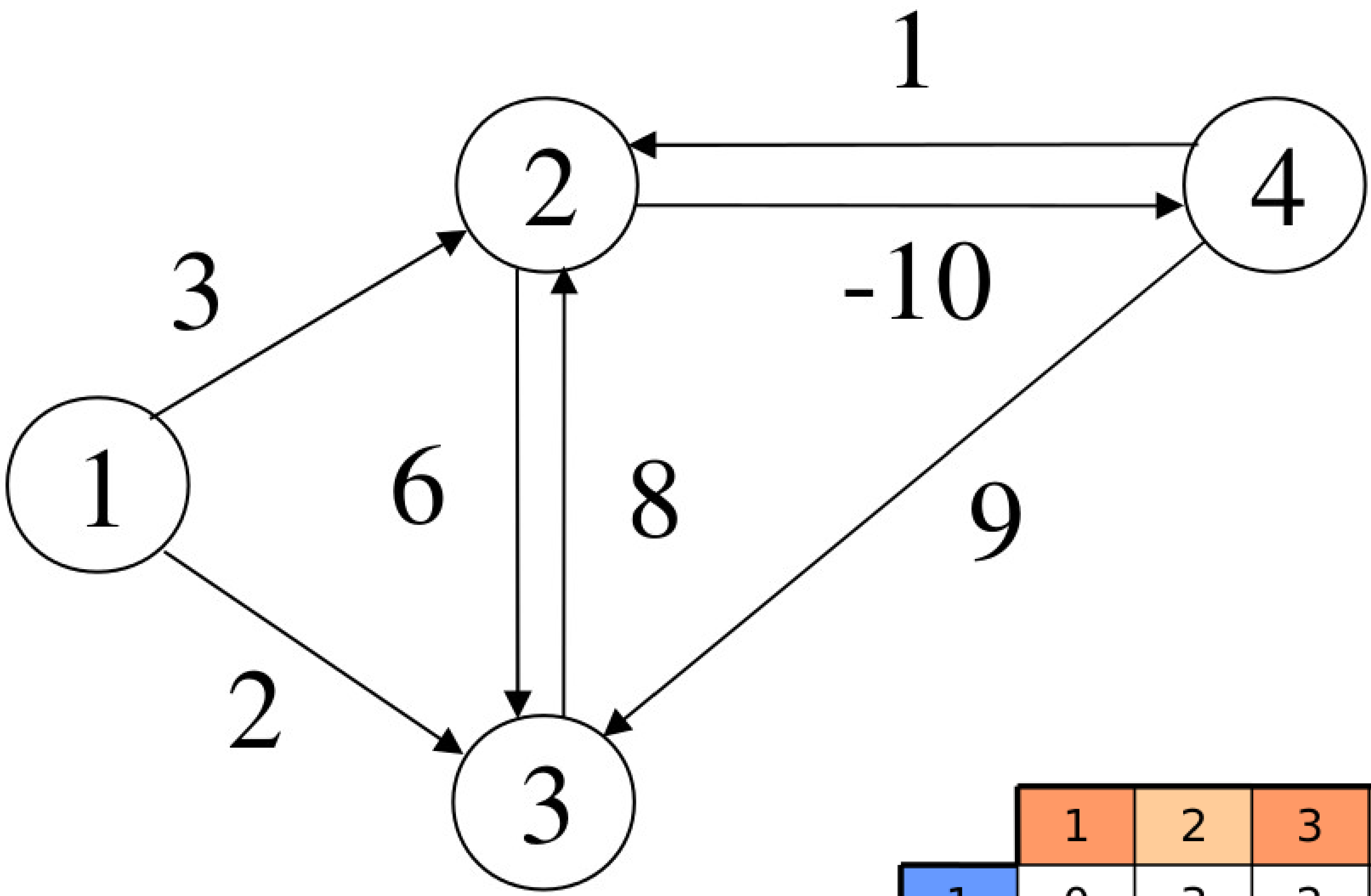
Алгоритмы на графах
II часть

Методы программирования — II семестр
Кирюхин В.А.
БК252

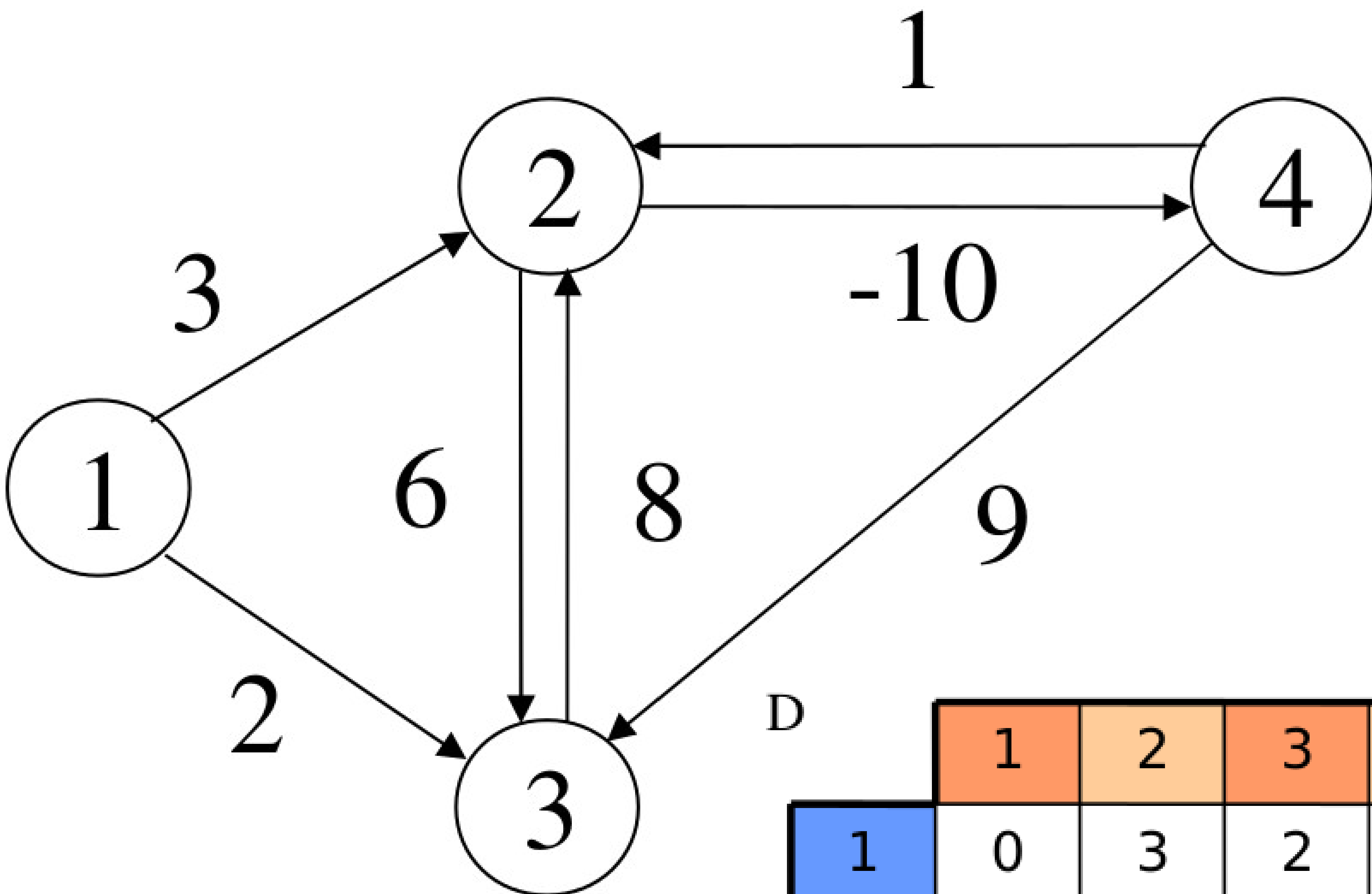
Алгоритм Флойда-Уоршелла
(при наличии ребер отрицательного веса)

Дан ориентированный граф **G**, содержащий ребра отрицательного веса.

Алгоритм Флойда-Уоршелла возвращает правильный результат, если граф не содержит циклов отрицательного веса. В противном случае сообщает, что имеется хотя бы один такой цикл.



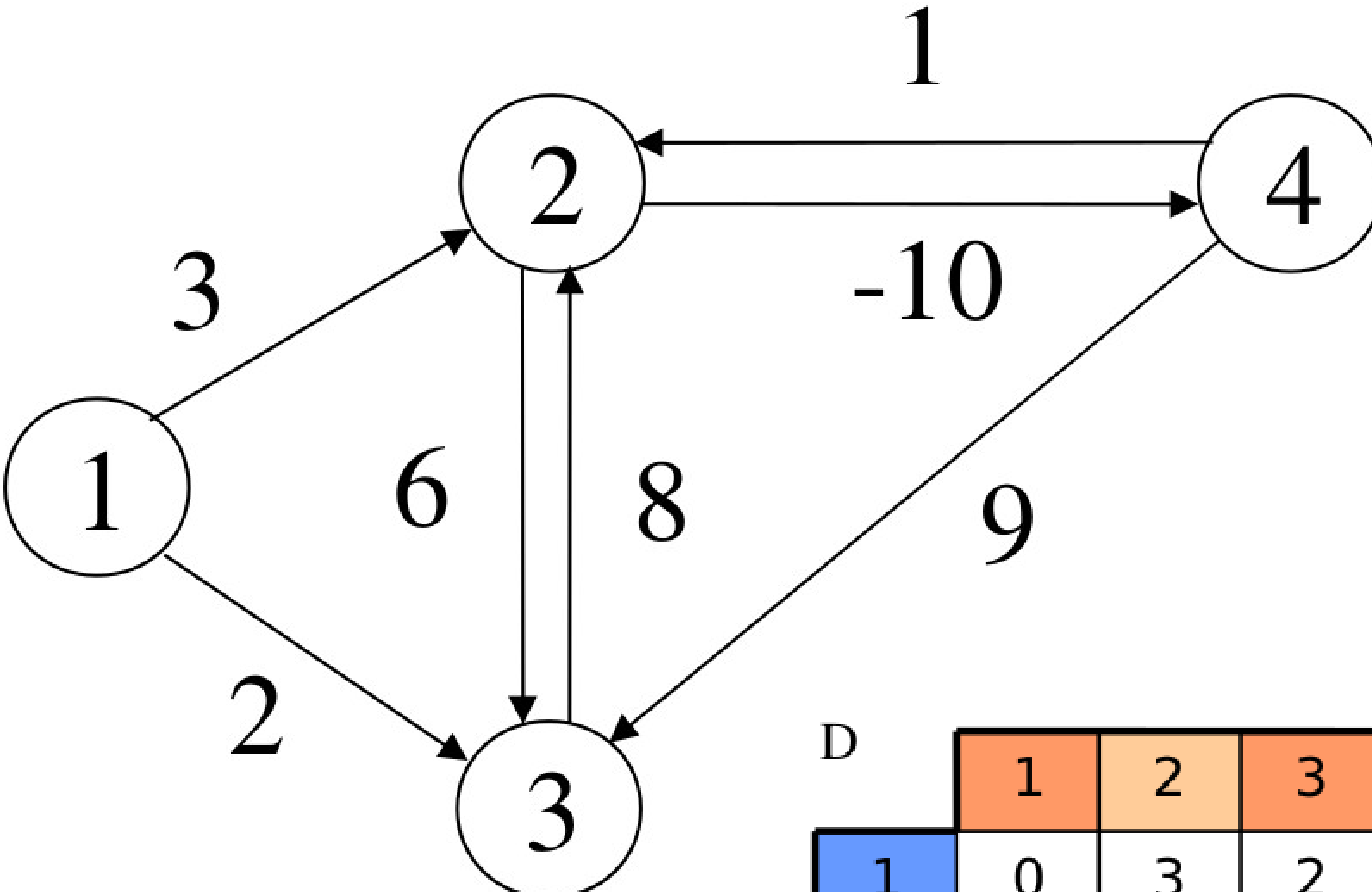
	1	2	3	4
1	0	3	2	∞
2	∞	0	6	-10
3	∞	8	0	∞
4	∞	1	9	0



Из 1 в 4 через 2

D

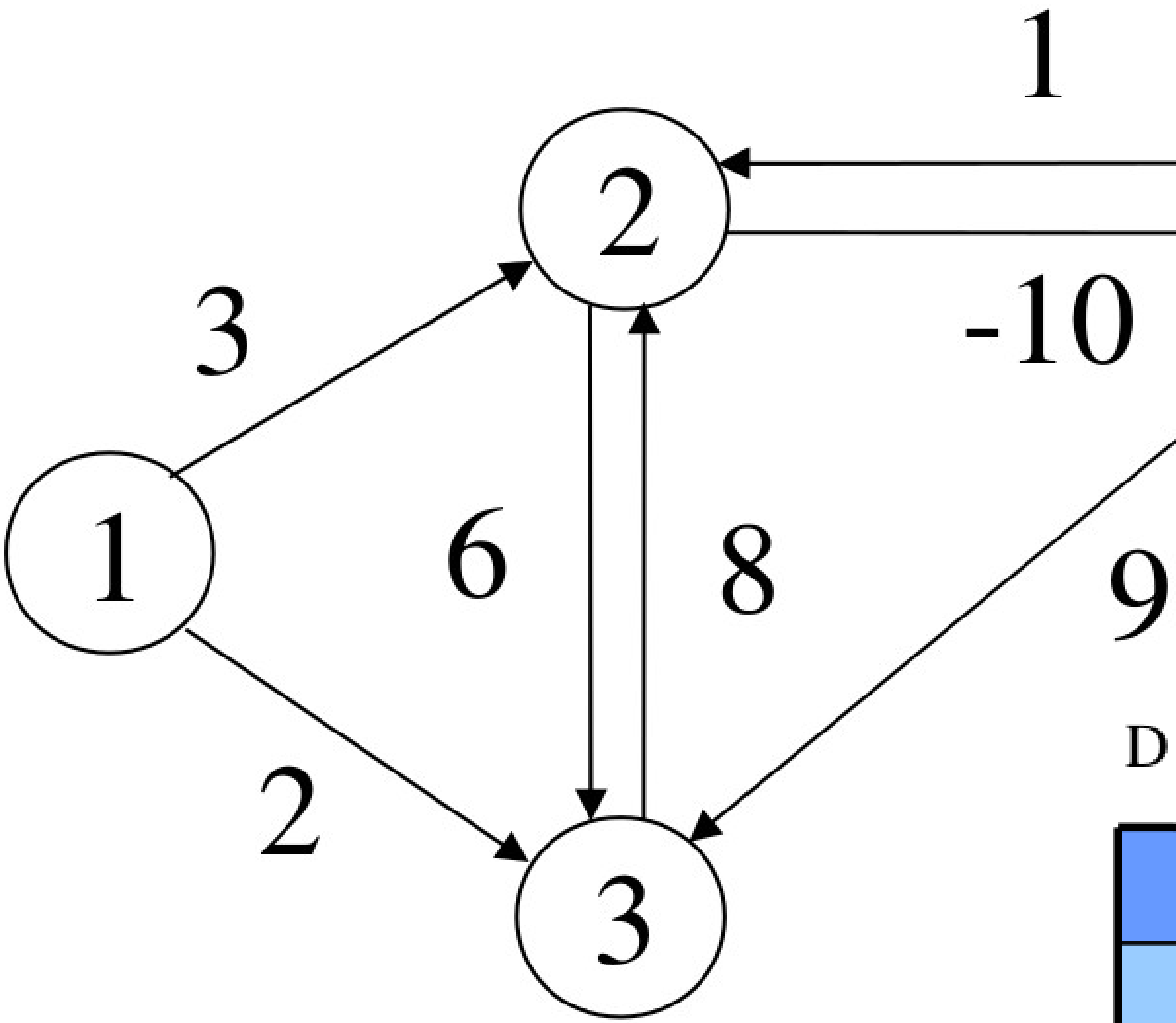
	1	2	3	4
1	0	3	2	-7
2	∞	0	6	-10
3	∞	8	0	∞
4	∞	1	9	0



D

	1	2	3	4
1	0	3	2	-7
2	∞	0	6	-10
3	∞	8	0	-2
4	∞	1	9	0

Из 3 в 4 через 2



D	1	2	3	4
1	0	3	2	-7
2	∞	0	6	-10
3	∞	8	0	-2
4	∞	1	7	-9

Из 4 в 3 через 2
Из 4 в 4 через 2

На главной диагонали отрицательное значение
→ существует цикл отрицательного веса

Алгоритм Флойда-Уоршелла

```
for k := 1 to
{
   for i := 1 to n
      for j := 1 to n
         D[i][j]:=min(D[i][j],
                      D[i][k]+D[k][j])
   for i := 1 to n
      if D[i][i] < 0 then
         exit
}
```

Топологическая сортировка

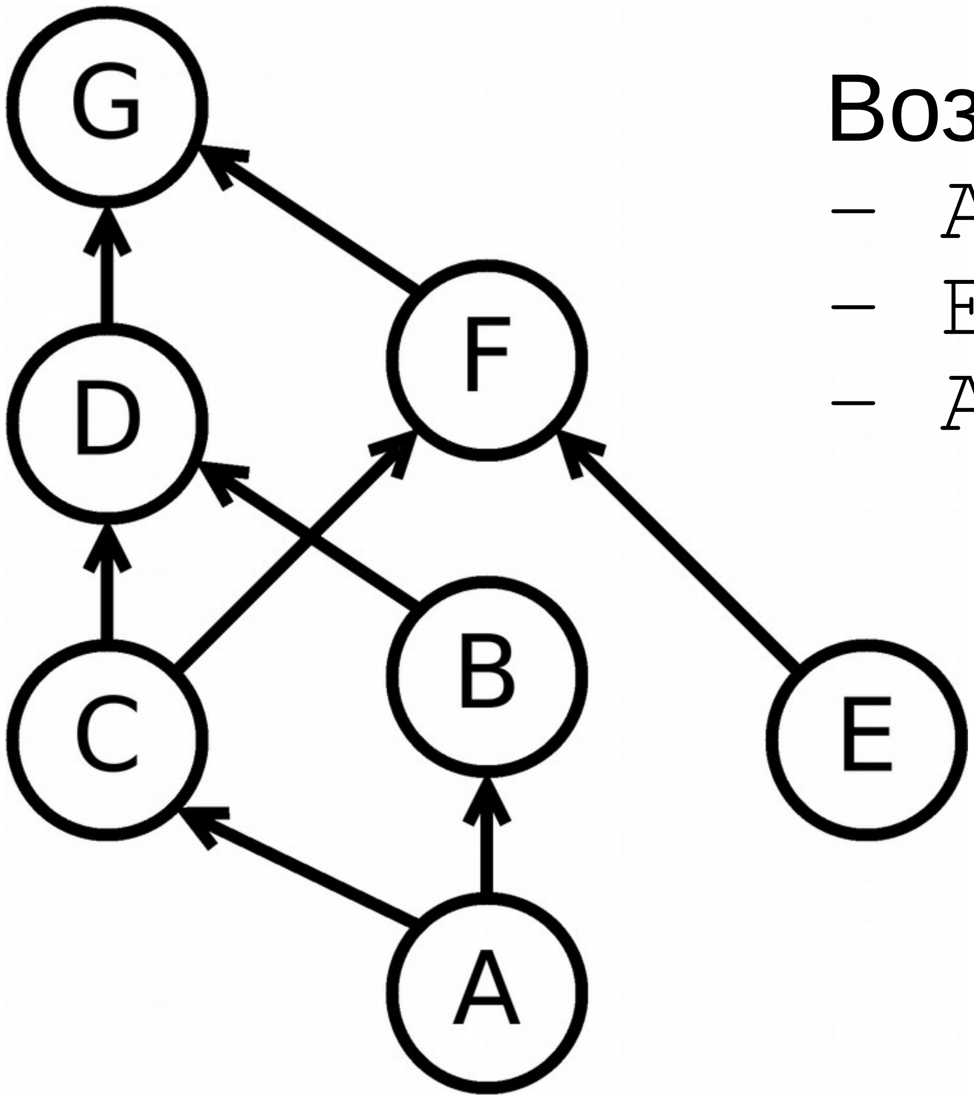
Дан ориентированный граф **G**.
Нужно пронумеровать вершины графа так, чтобы для каждого ребра графа **(v,v')** было выполнено **v** < **v'**

Т.е. упорядочить вершины **G** согласно частичному порядку, заданному ребрами.

Топологическая сортировка может быть
НЕ единственной.

Топологической сортировки может НЕ существовать

Топологическая сортировка

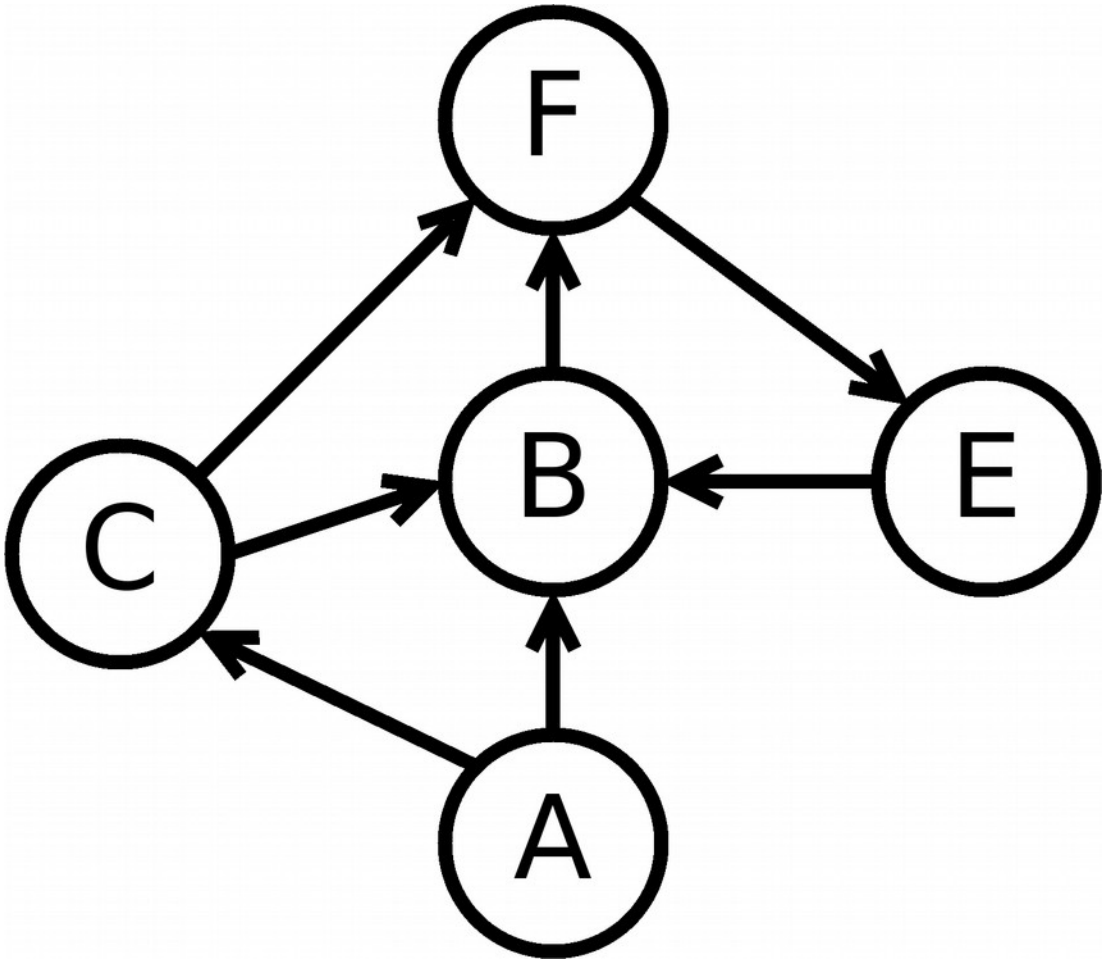


Возможные варианты:

- A,E,B,C,F,D,G
- E,A,C,B,D,F,G
- A,C,B,D,E,F,G

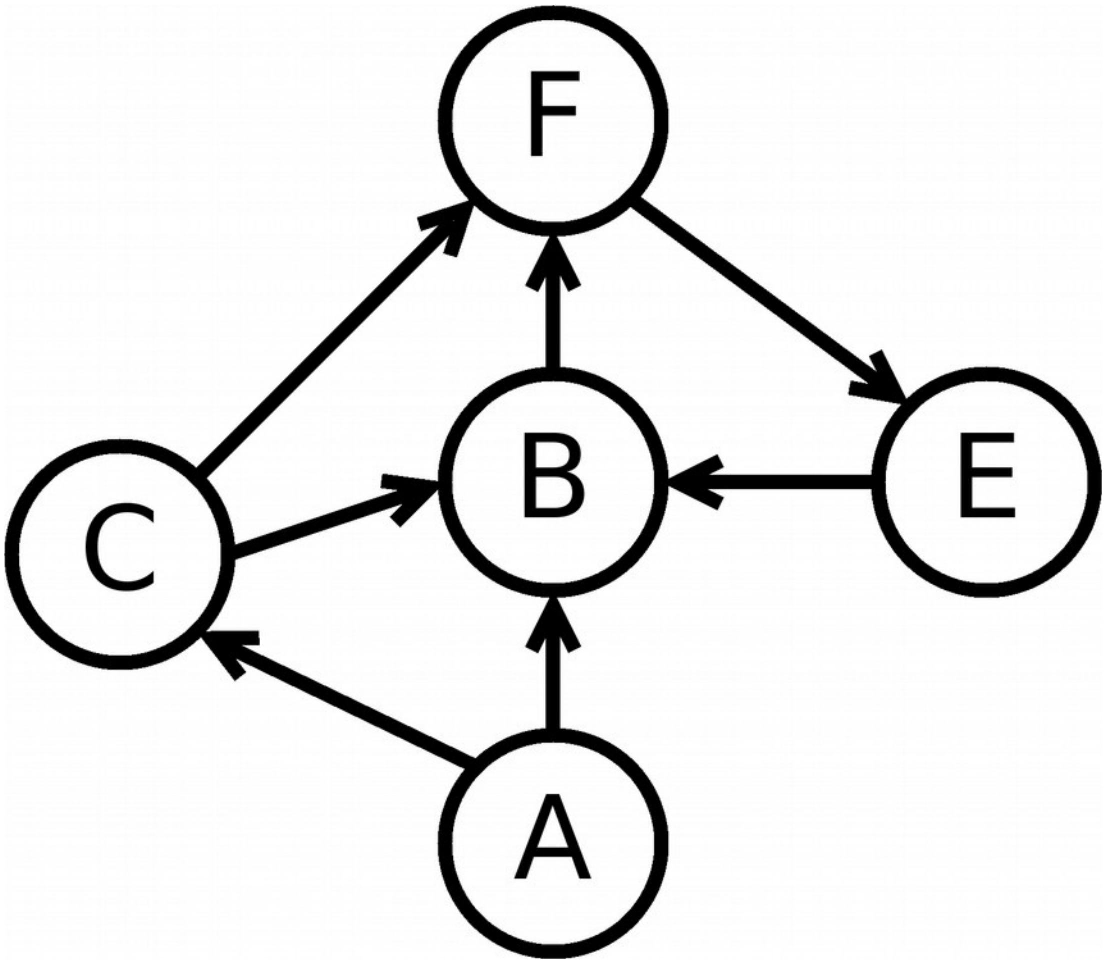
Топологическая сортировка

НЕ существует в случае:



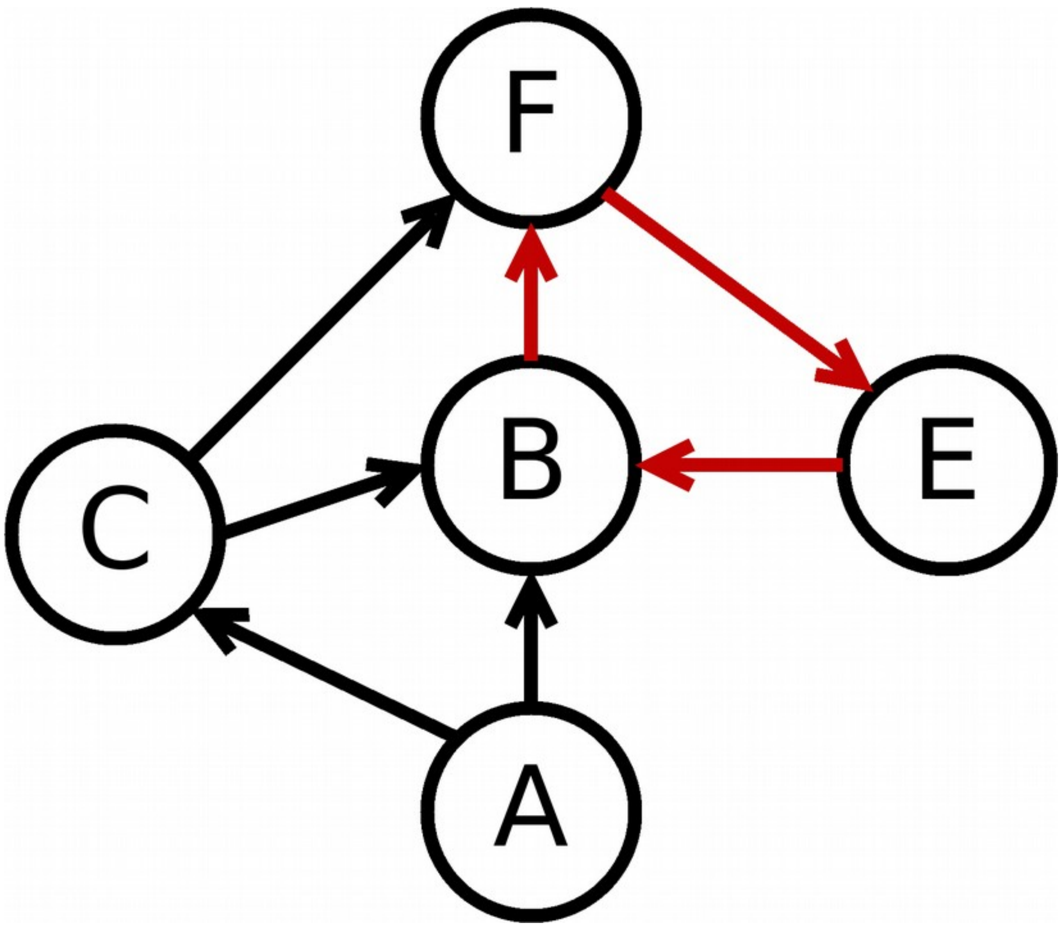
Топологическая сортировка

НЕ существует в случае:



Топологическая сортировка

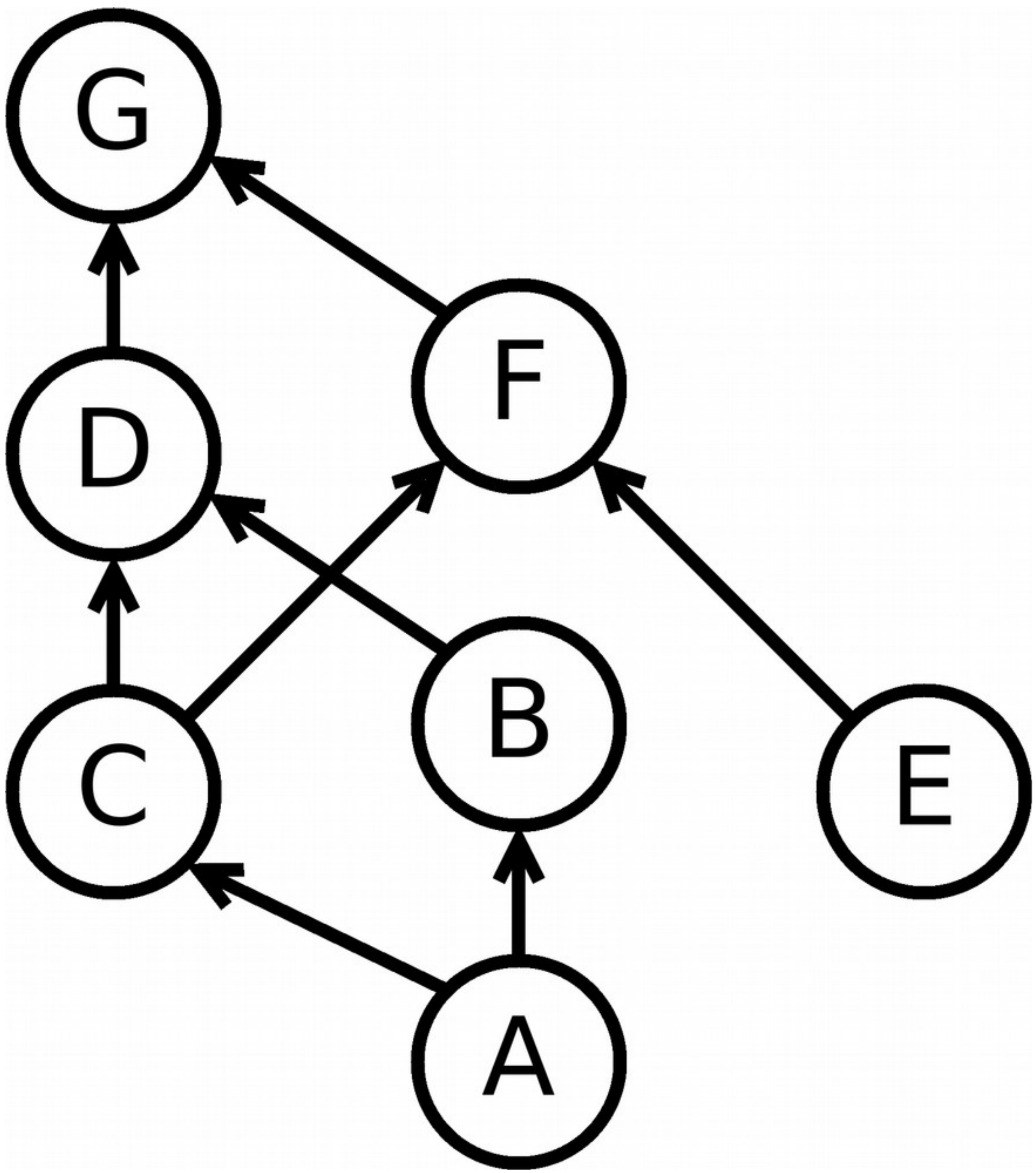
НЕ существует в случае:



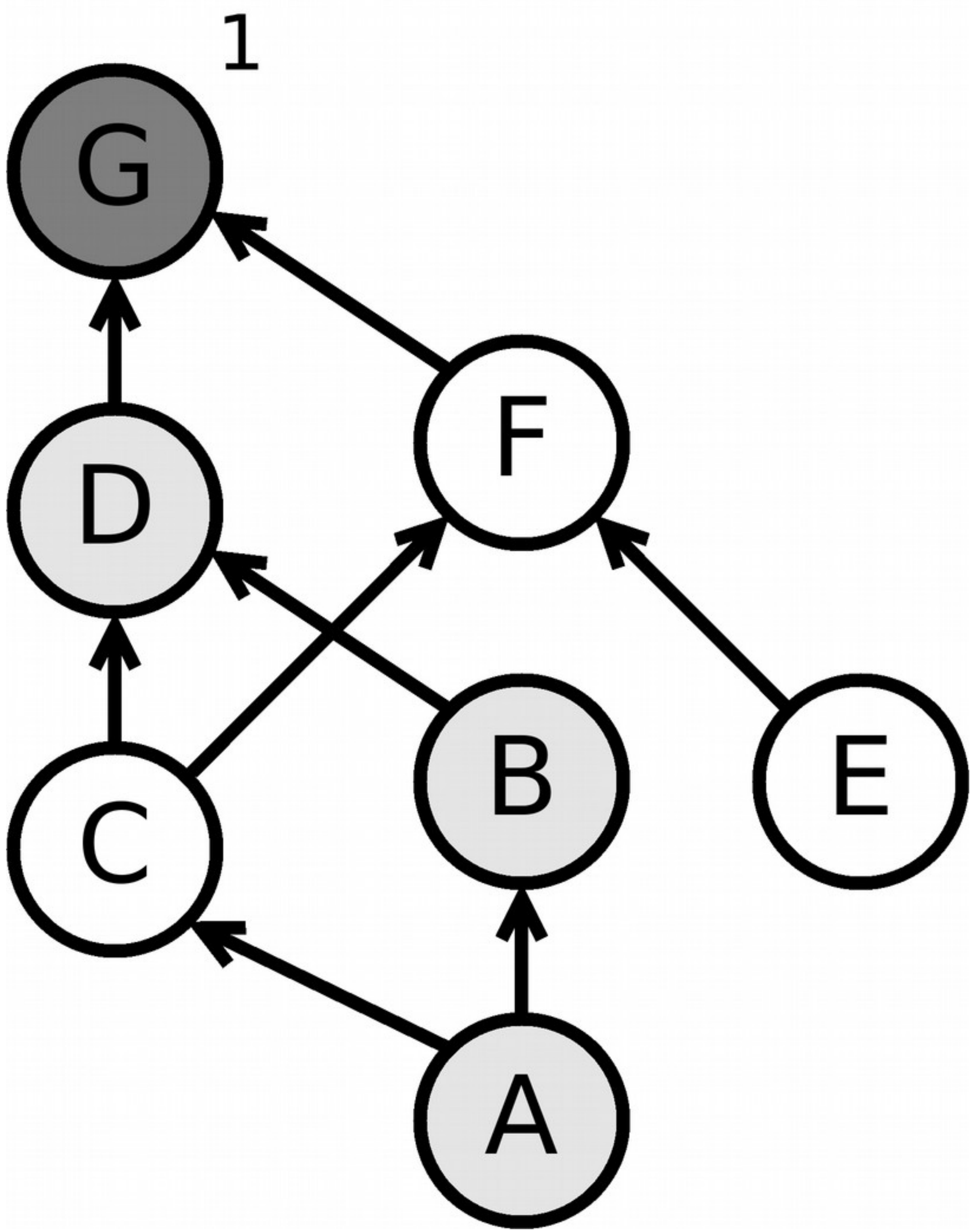
Топологическая сортировка
Алгоритм Тарьяна

Идея: будем выполнять поиск в глубину из каждой непосещенной ранее вершины и запоминать время выхода из каждой. Отсортируем вершины по времени выхода от большего к меньшему.

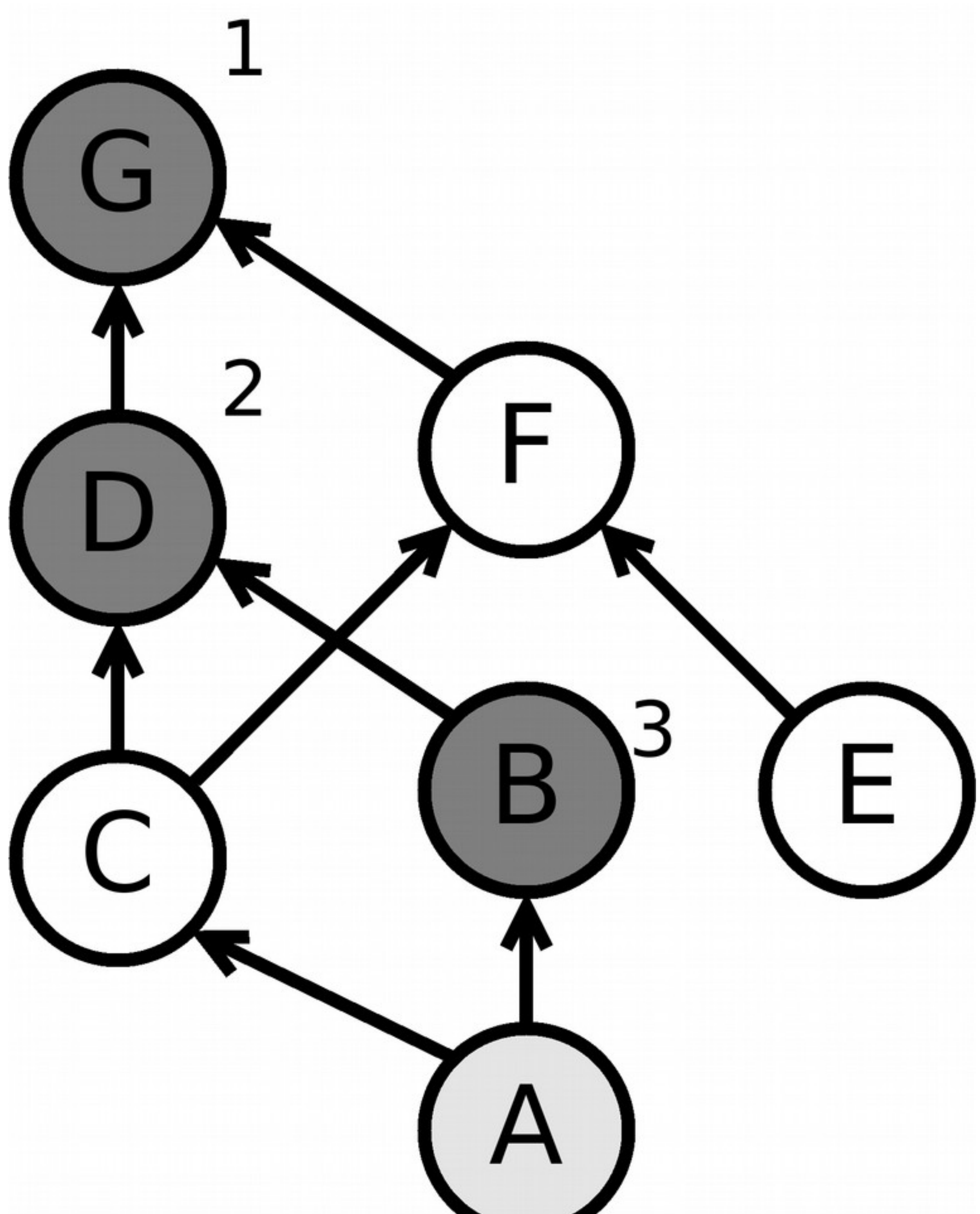
Алгоритм Тарьяна



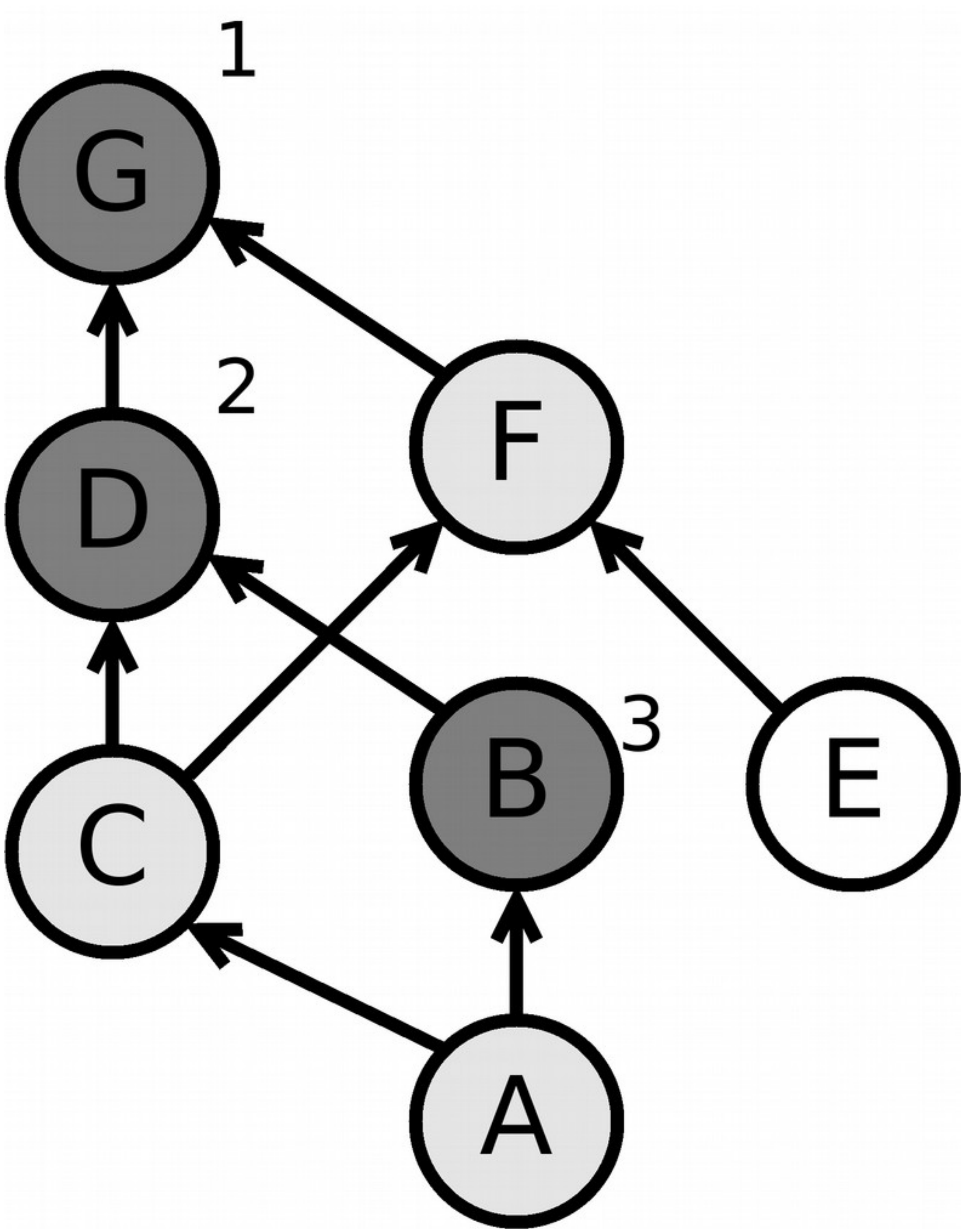
Алгоритм Тарьяна



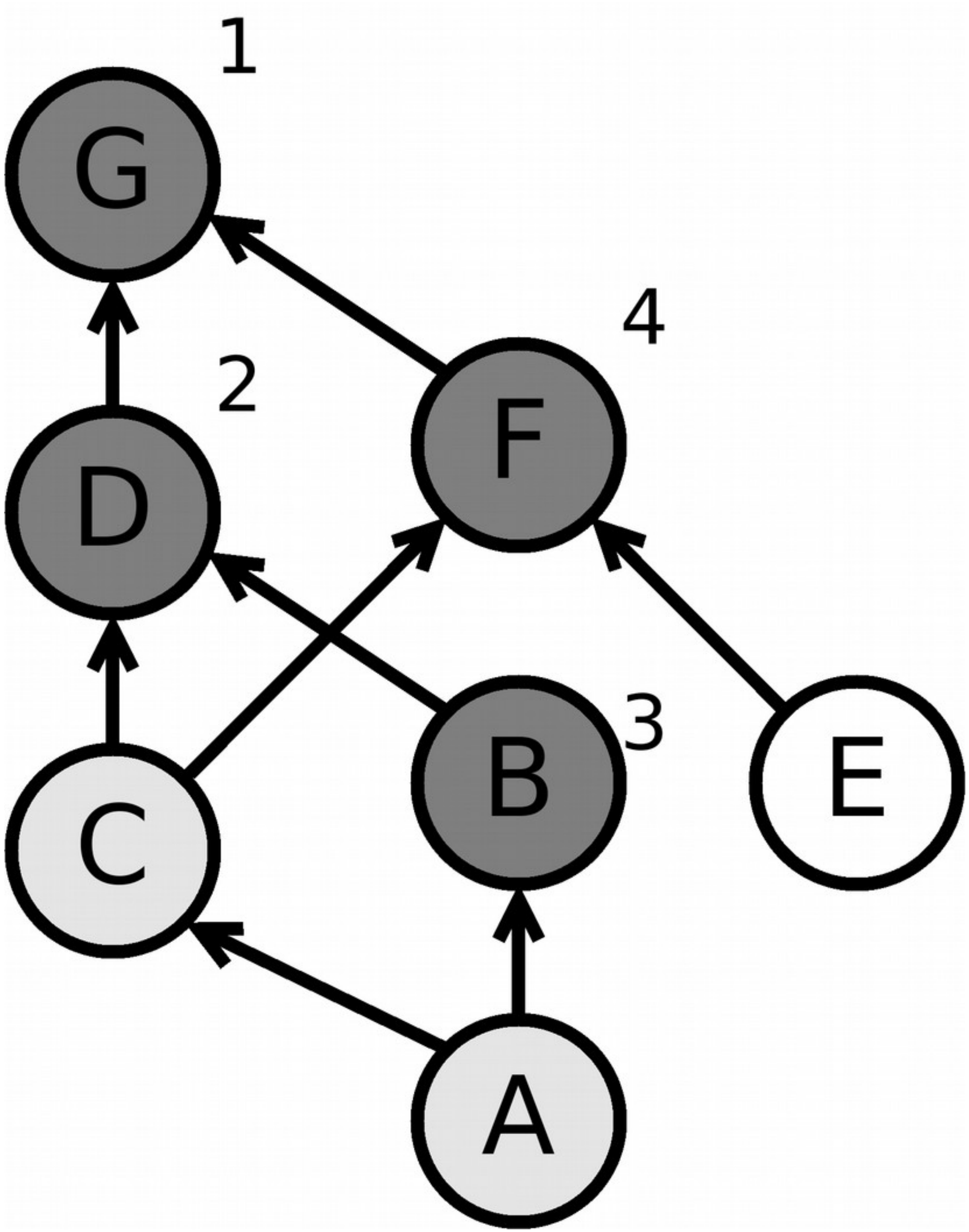
Алгоритм Тарьяна



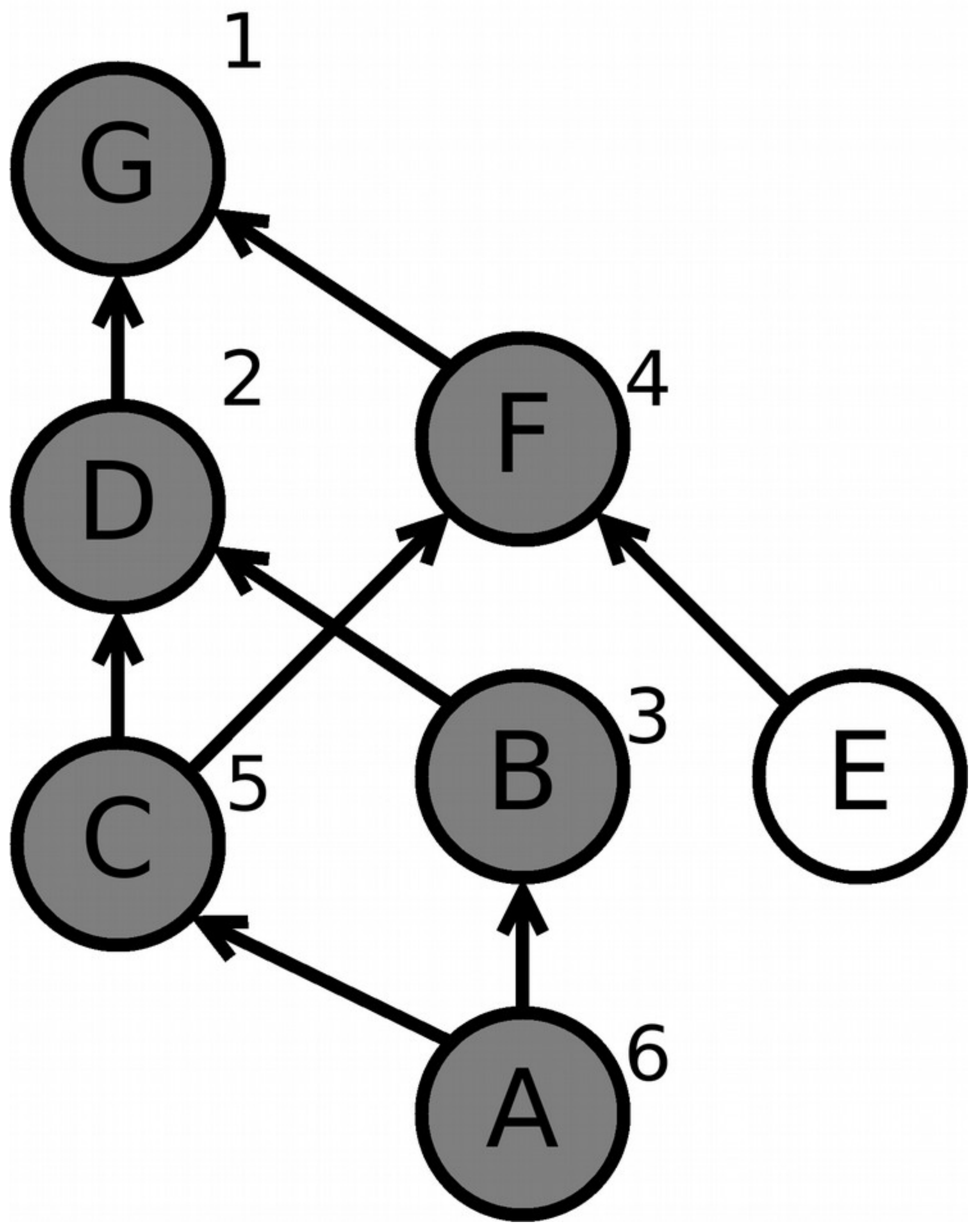
Алгоритм Тарьяна



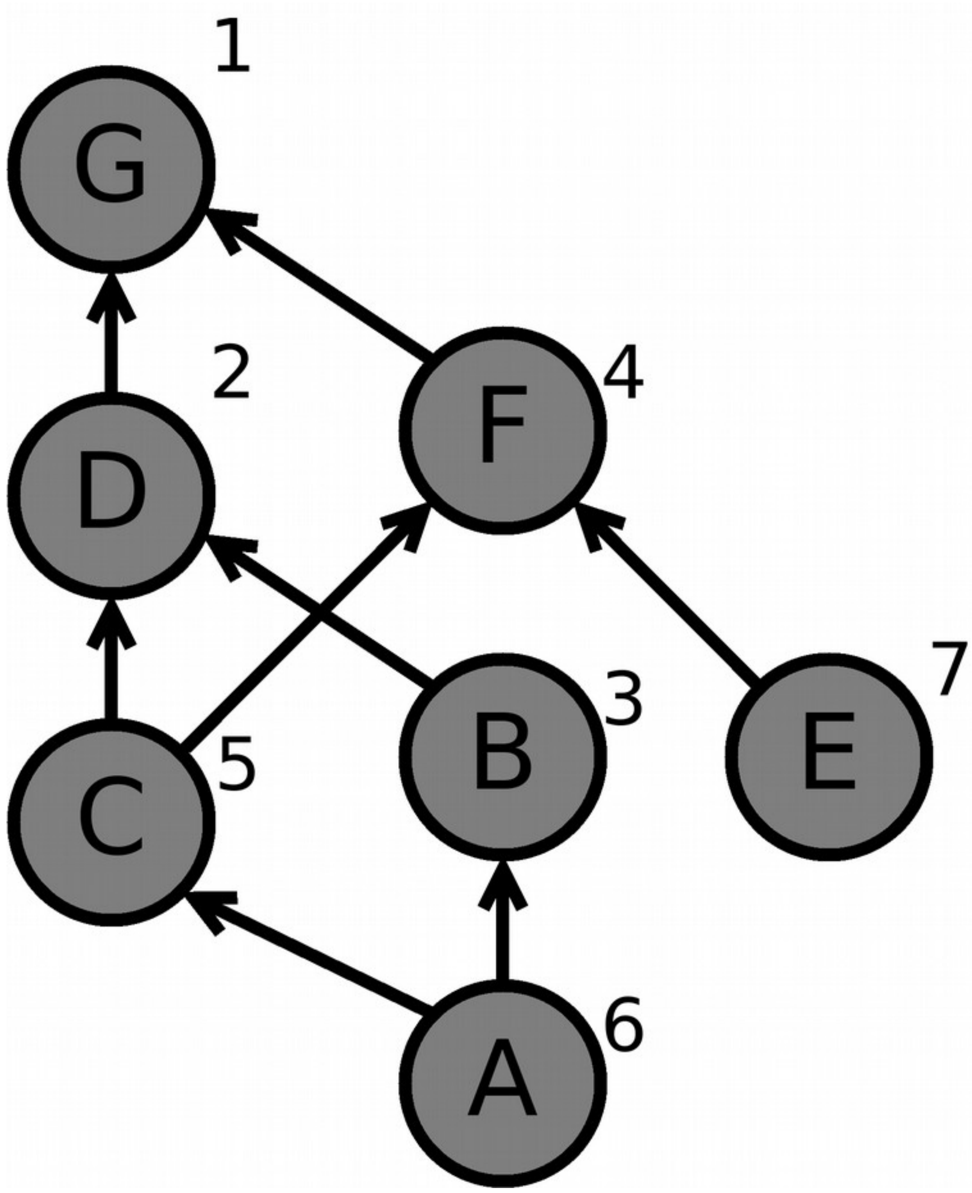
Алгоритм Тарьяна



Алгоритм Тарьяна



Алгоритм Тарьяна



Топологическая сортировка: E, A, C, F, B, D, G

Алгоритм Тарьяна

```
Visit[n] — массив посещений
Stack — последовательность вершин - результат

function dfs_inv(x)
   Visit[x] := true
   Для всех y смежных с x
      Если Visit[y] == false
         dfs_inv(y)
   x → Stack
-----------------------------
Для всех v из G:
   Visit[v] := false
Для всех v из G:
   Если Visit[v] == false
      dfs_inv(v)
```

Поиск Эйлерова цикла в графе

Эйлеров цикл — цикл графа, проходящий через каждое ребро ровно один раз.

В неориентированном графе эйлеров цикл существует тогда и только тогда, когда граф **связный** и **степени всех его вершин четные**.

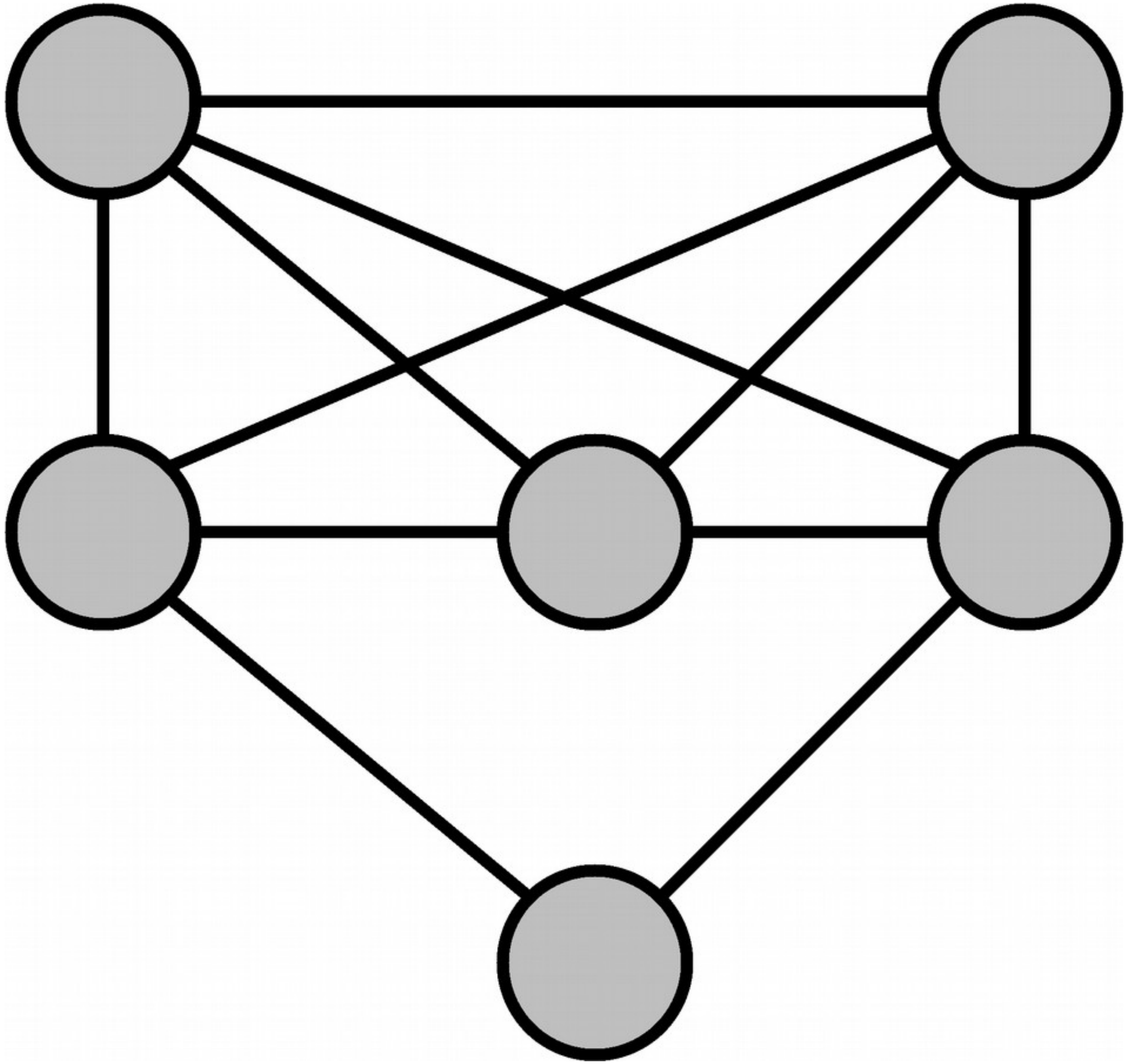
Алгоритм Флёри

Идея:
- начинаем с произвольной вершины **s**
- выбираем произвольное ребро **(s,v)**
- заносим **(s,v)** в результат
- удаляем **(s,v)** графа
- переходим в вершину **v**

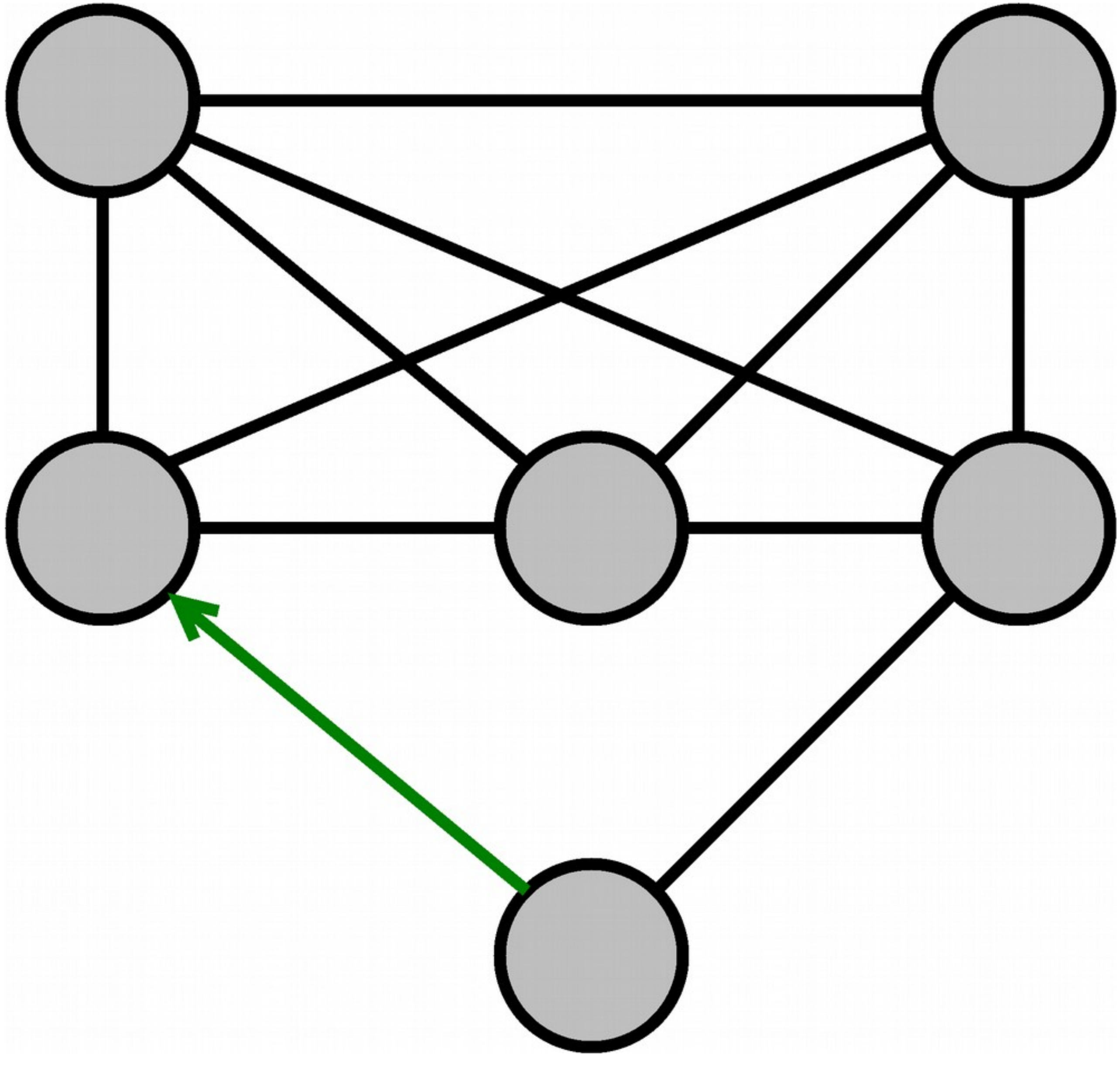
На каждом шаге алгоритма выбираем ребро инцидентное текущей вершине, мост выбираем только в том случае, если других вариантов нет. Заносим выбранное ребро в результат и удаляем из графа.

Мост — ребро, удаление которого увеличивает число компонент связности.

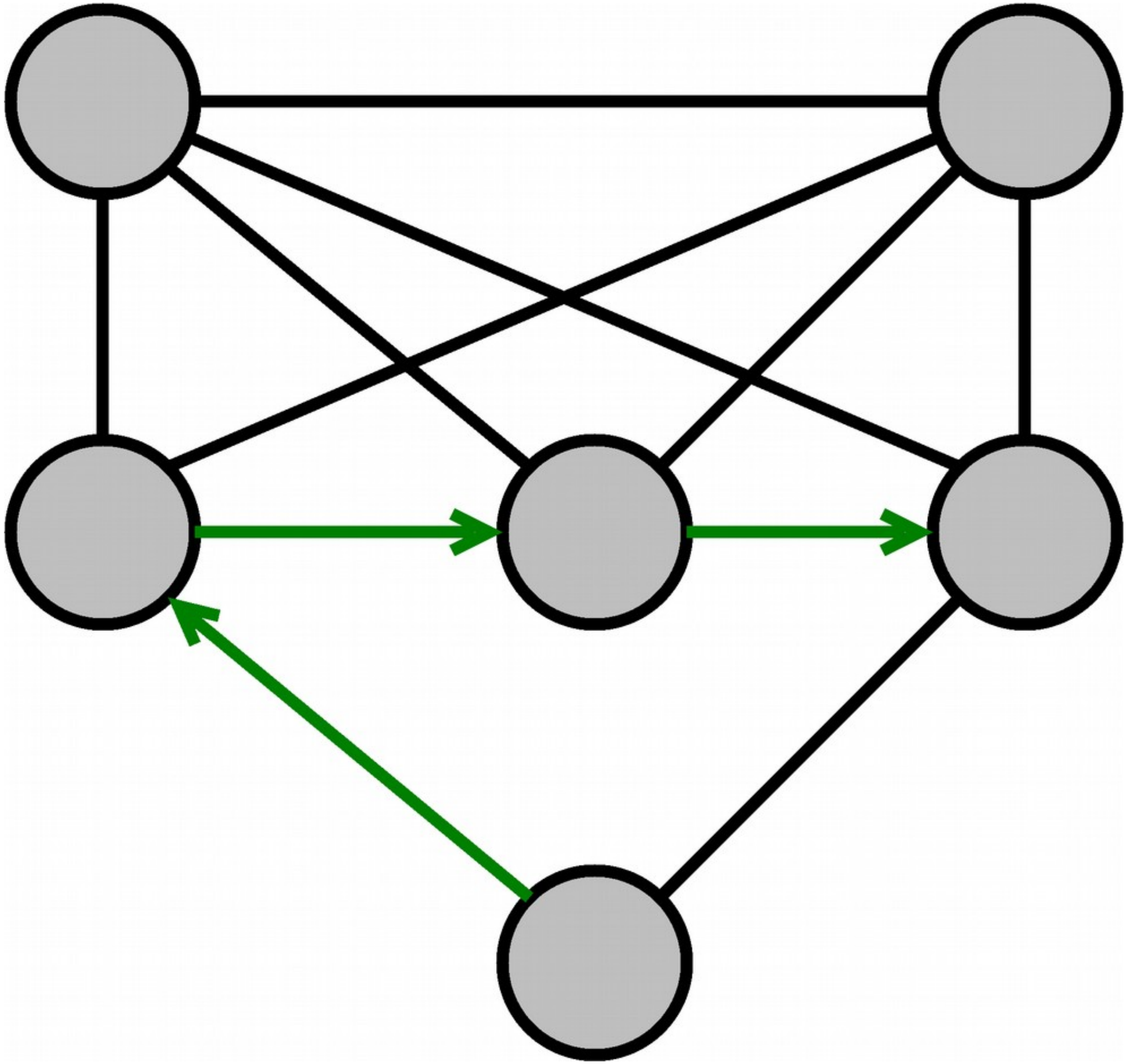
Алгоритм Флёри



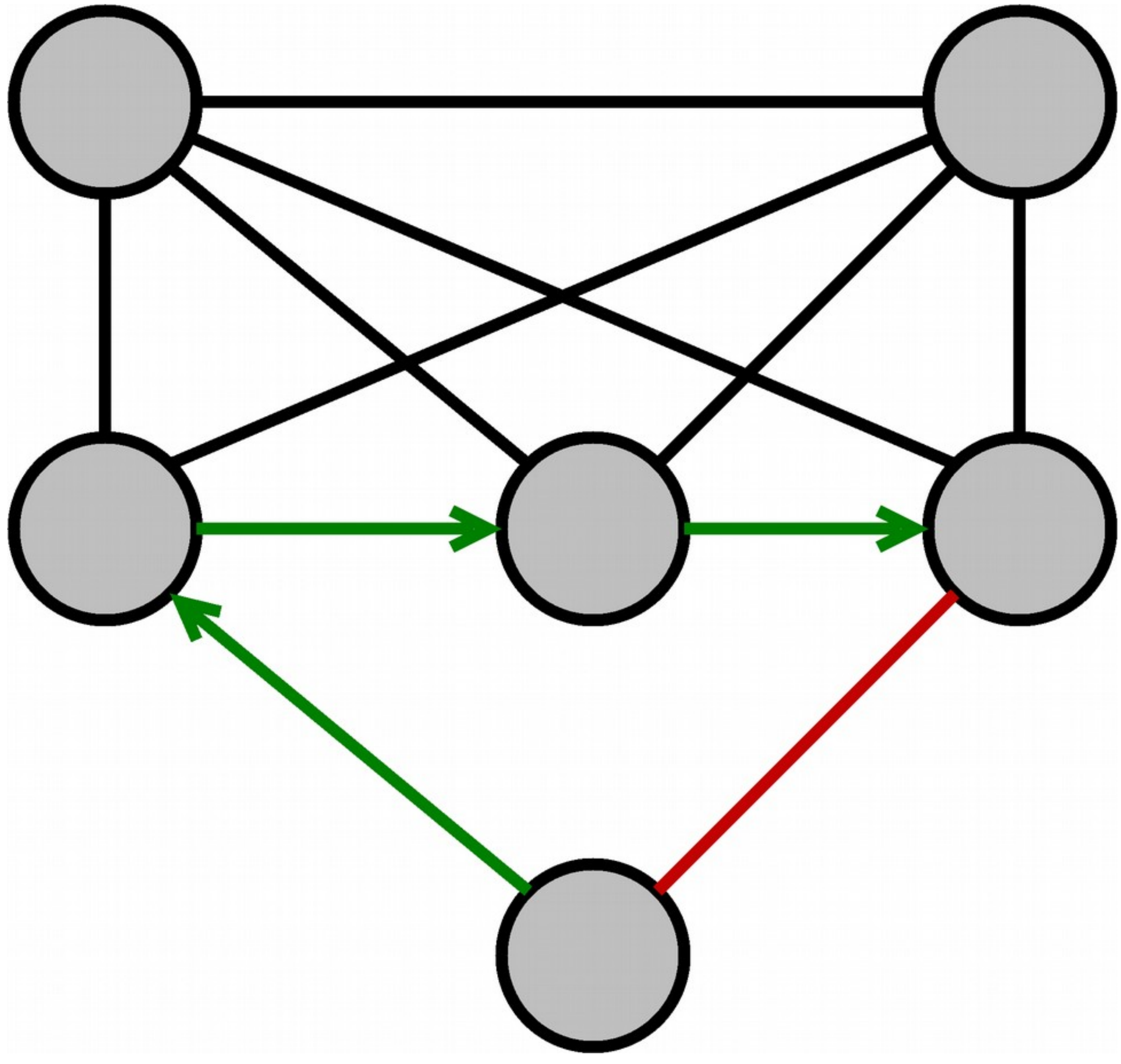
Алгоритм Флёри



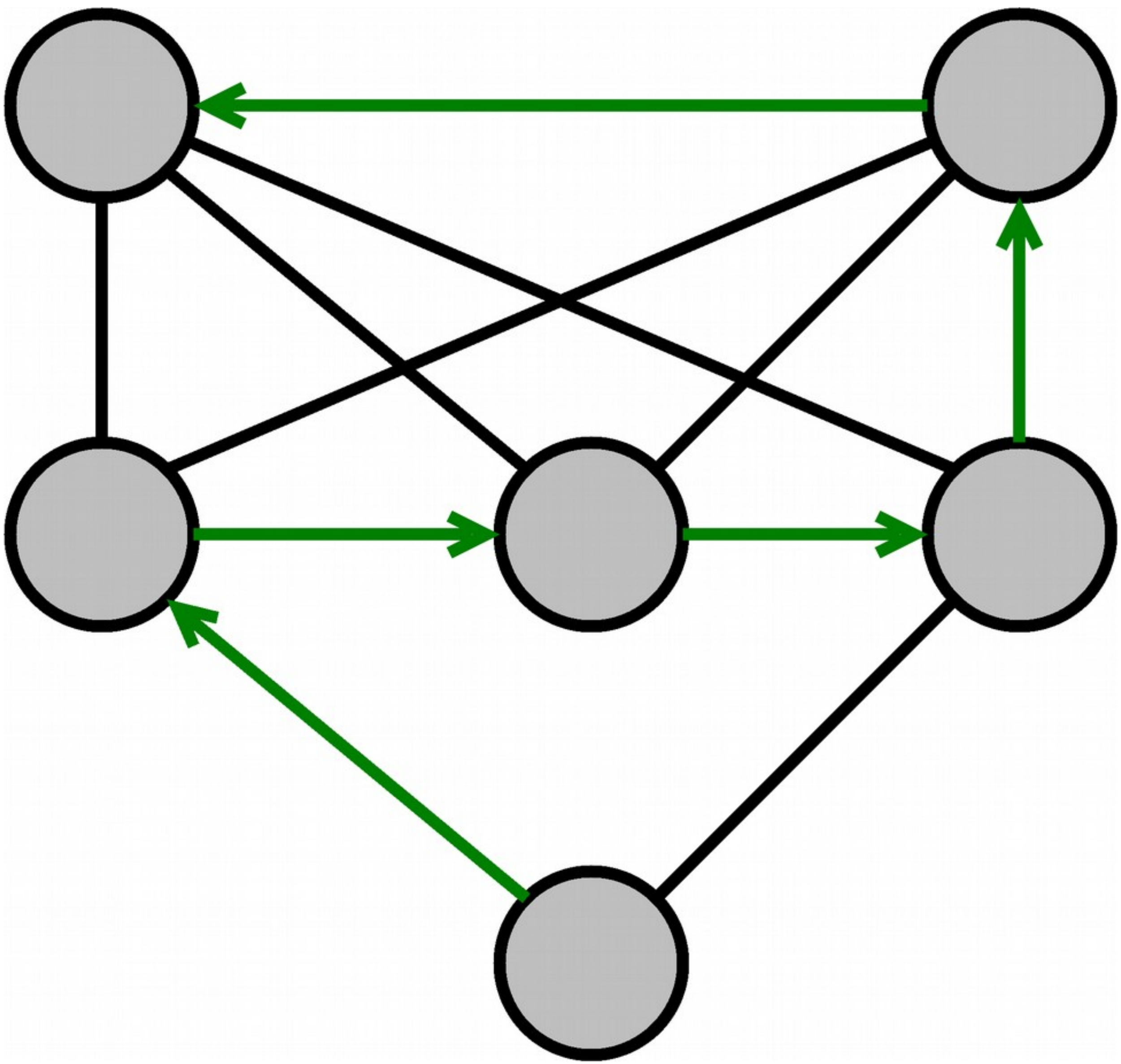
Алгоритм Флёри



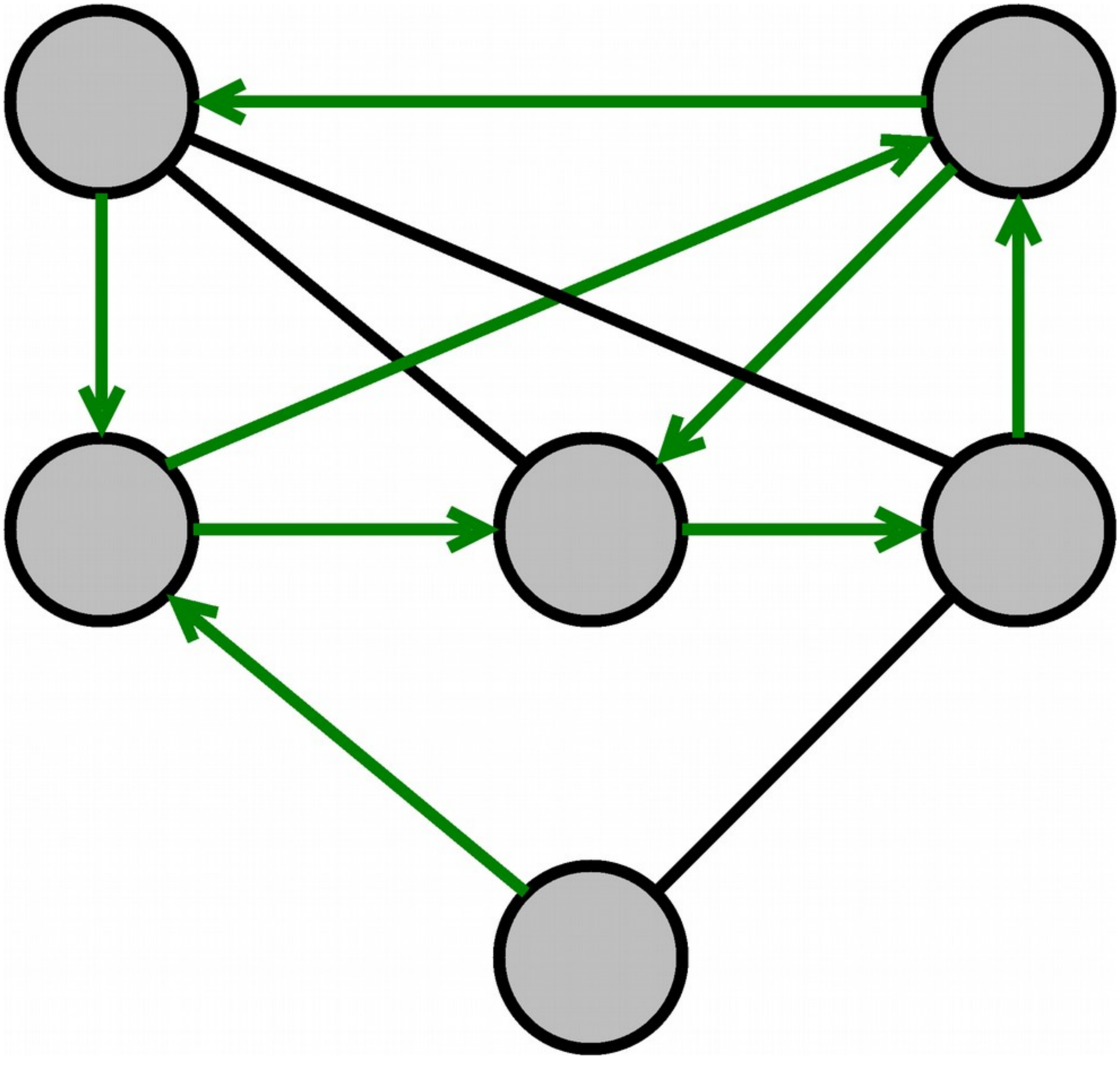
Алгоритм Флёри



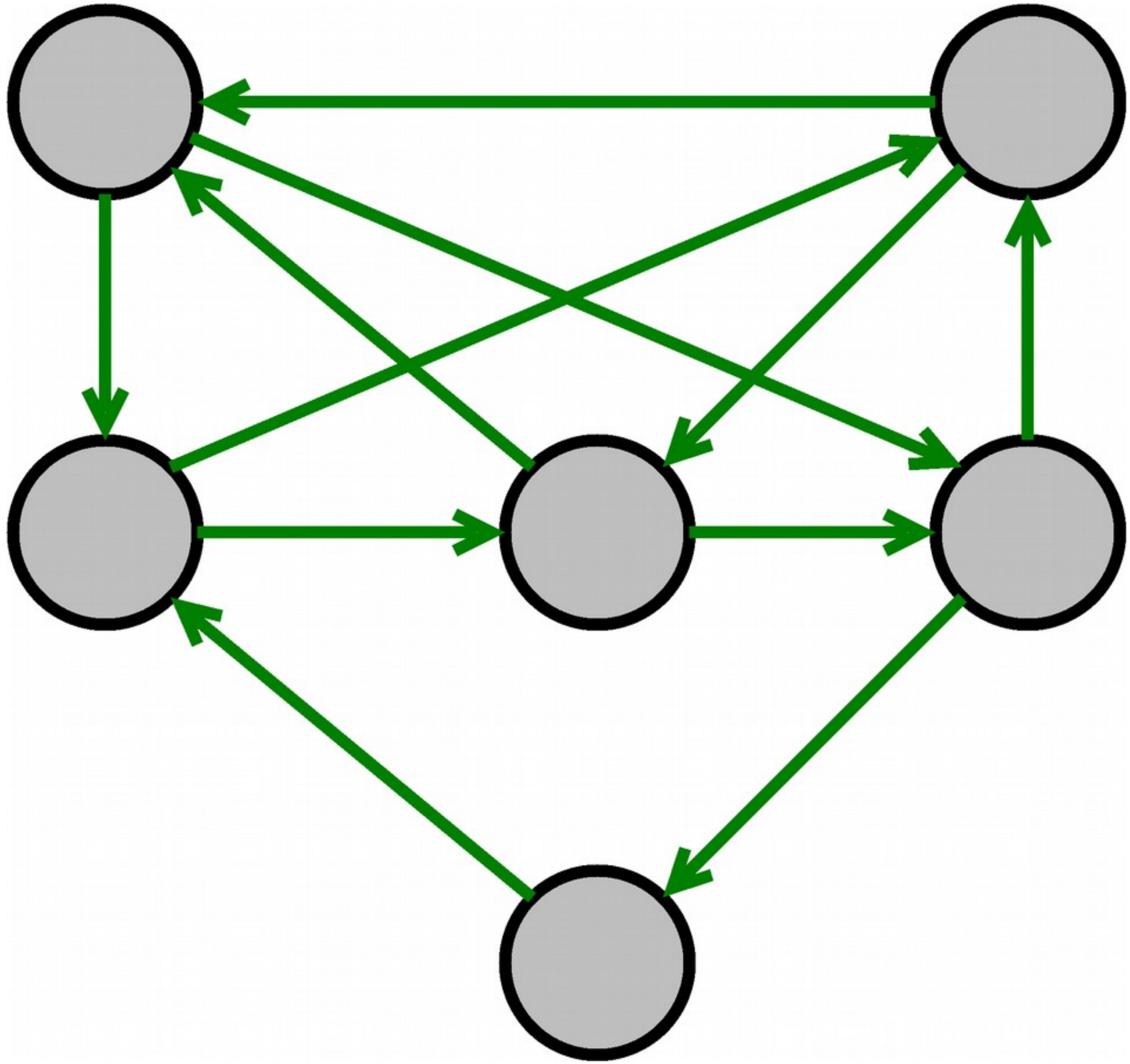
Алгоритм Флёри



Алгоритм Флёри



Алгоритм Флёри



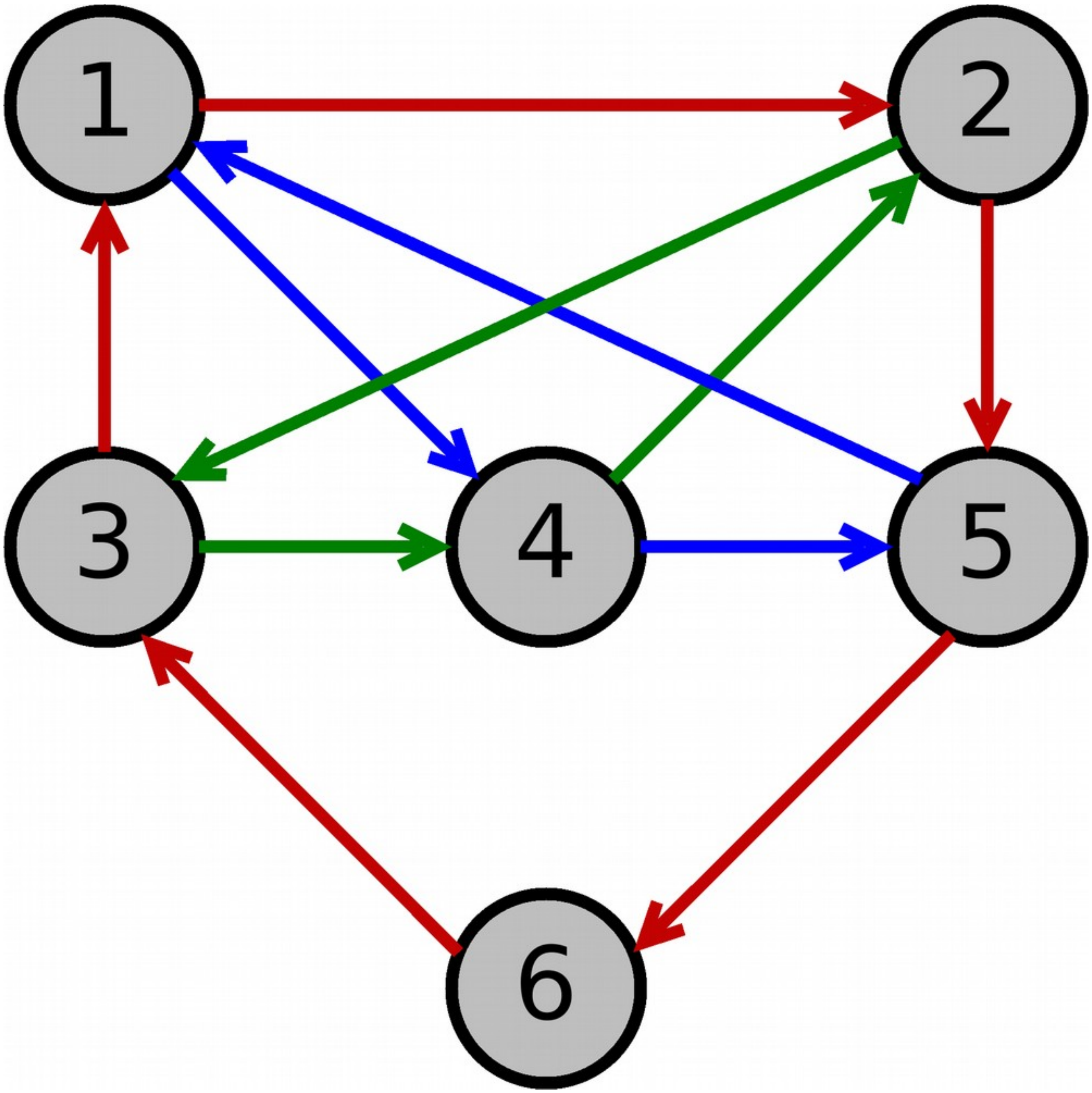
Алгоритм Флёри

```
Result — последовательность ребер
s — произвольная стартовая вершина

v := s
while |G.edges| > 0
    выбрать e = (v,v'): e не мост или единственное
        удалить из G ребро e
        e → Result
        v := v'
```

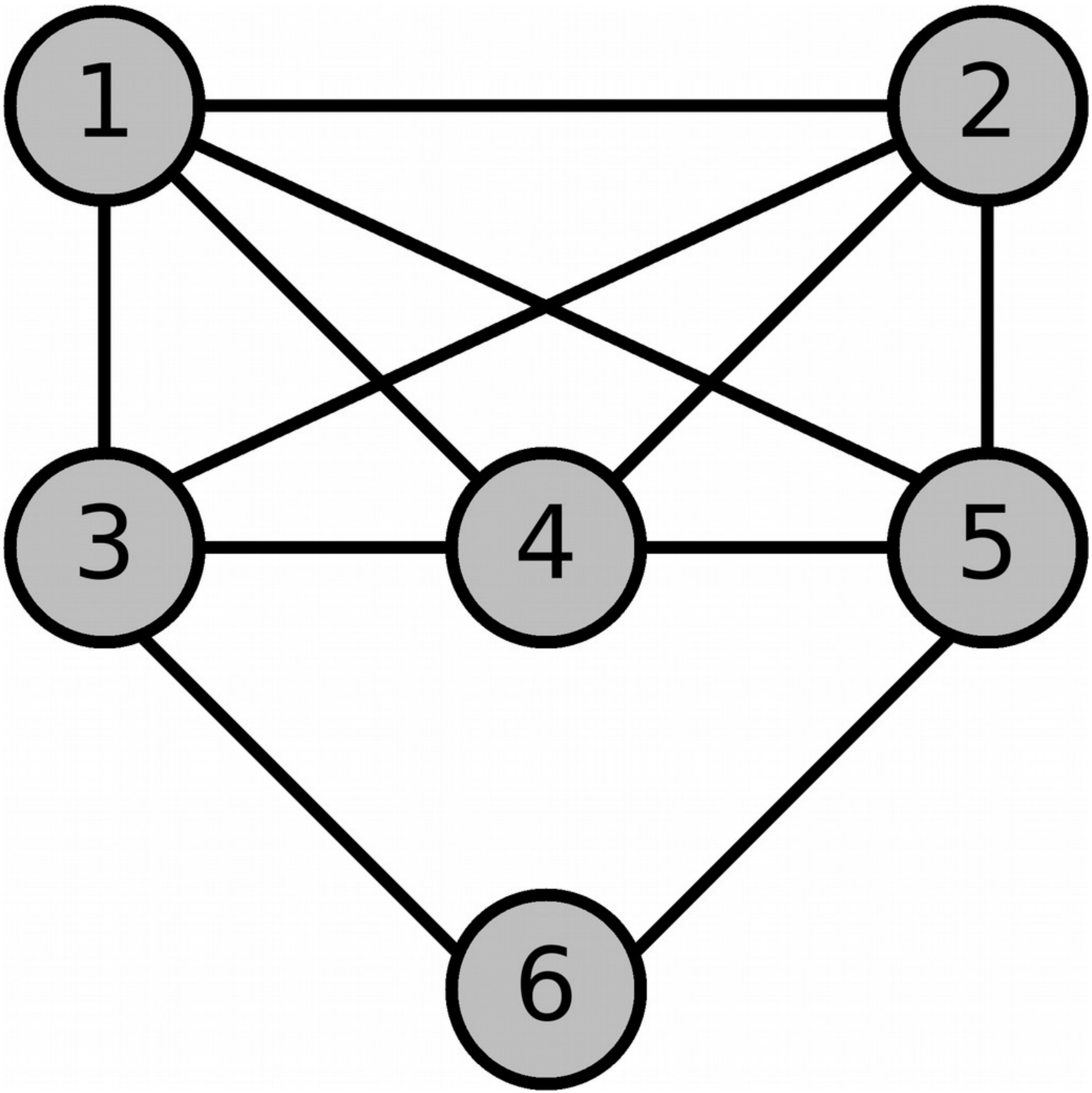
Алгоритм на основе циклов

Идея: найти все простые циклы графа и объединить их в один эйлеров цикл.

Алгоритм на основе циклов

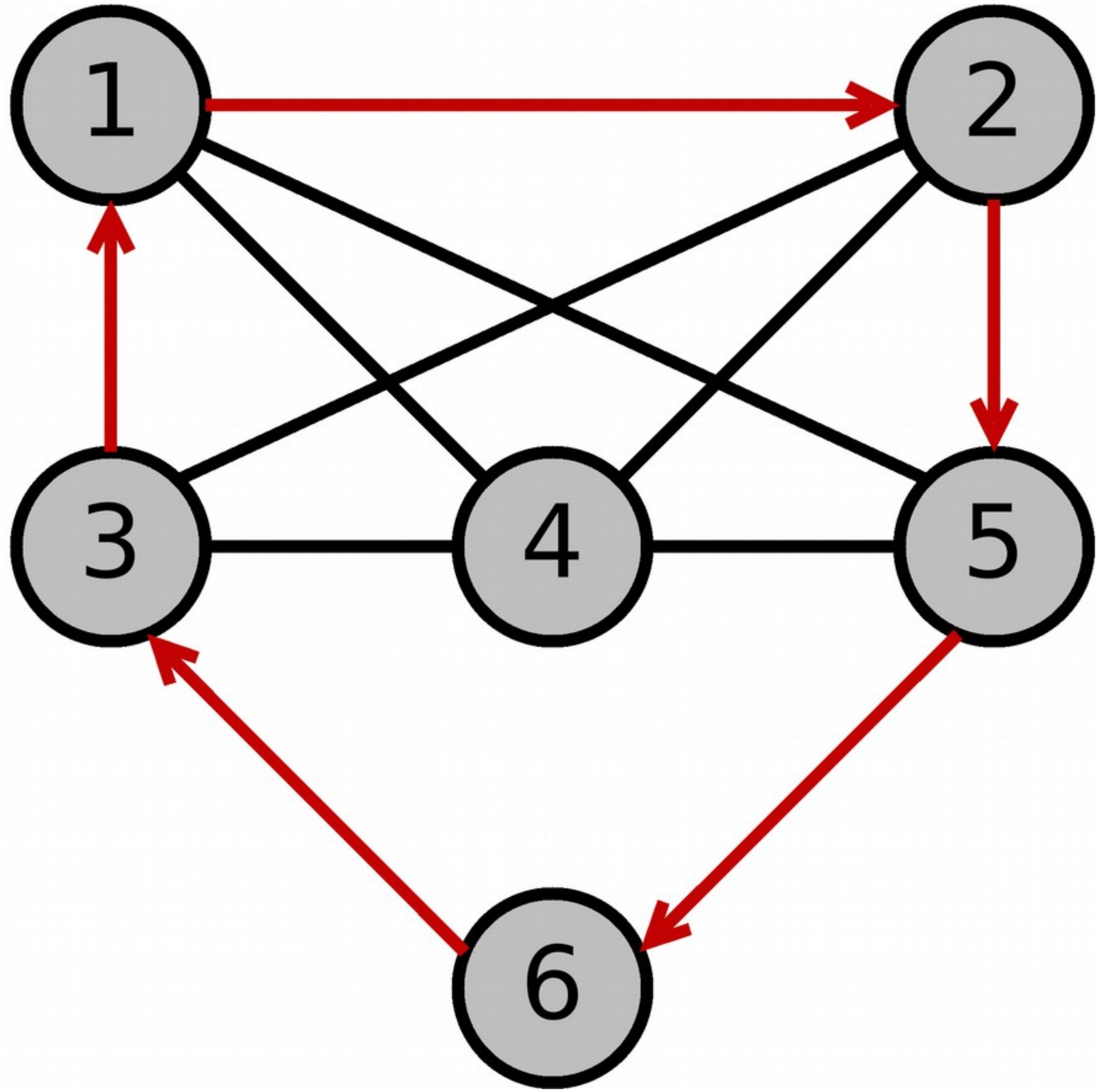


Алгоритм на основе циклов

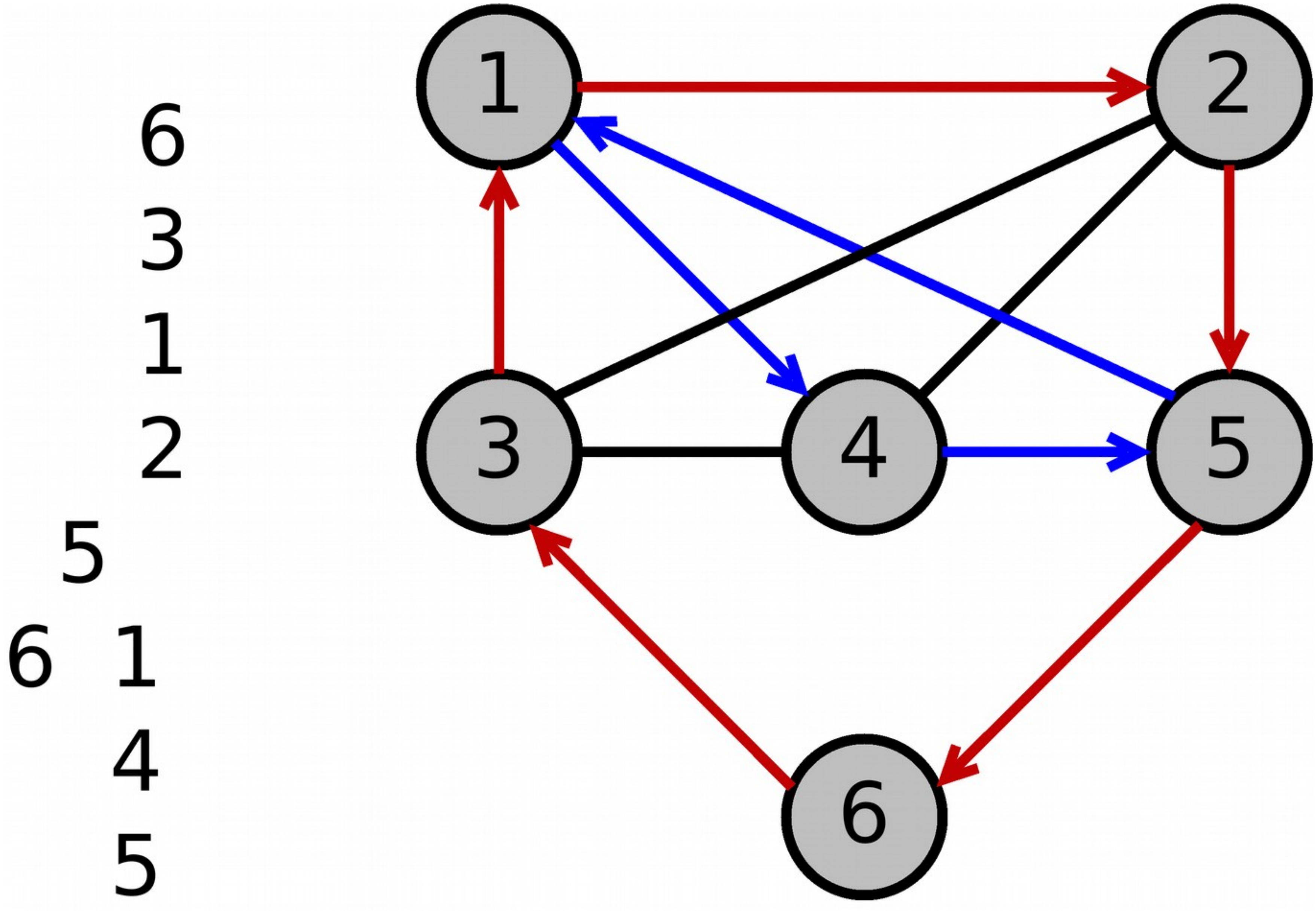


Алгоритм на основе циклов

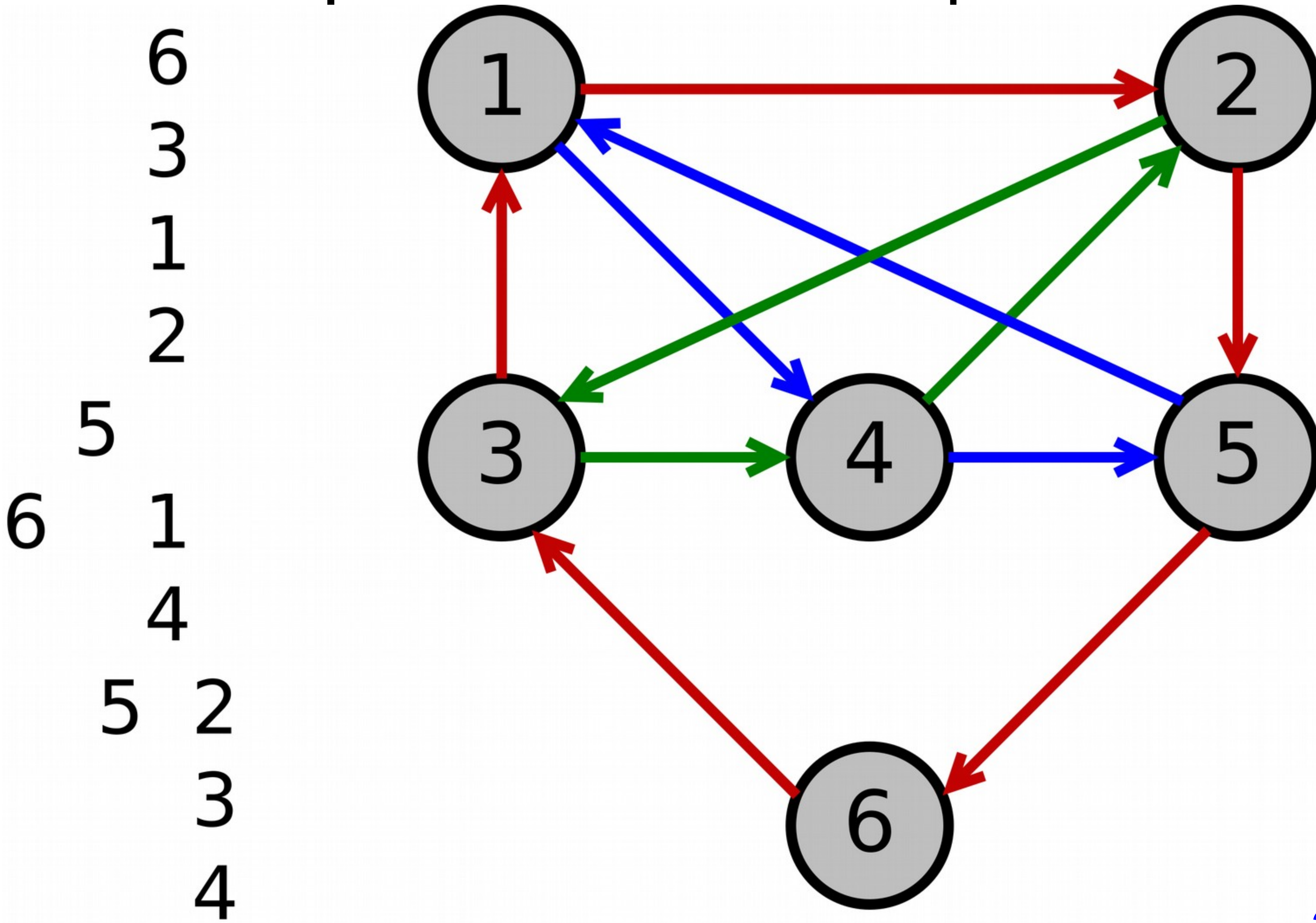
6
3
1
2
5
6



Алгоритм на основе циклов

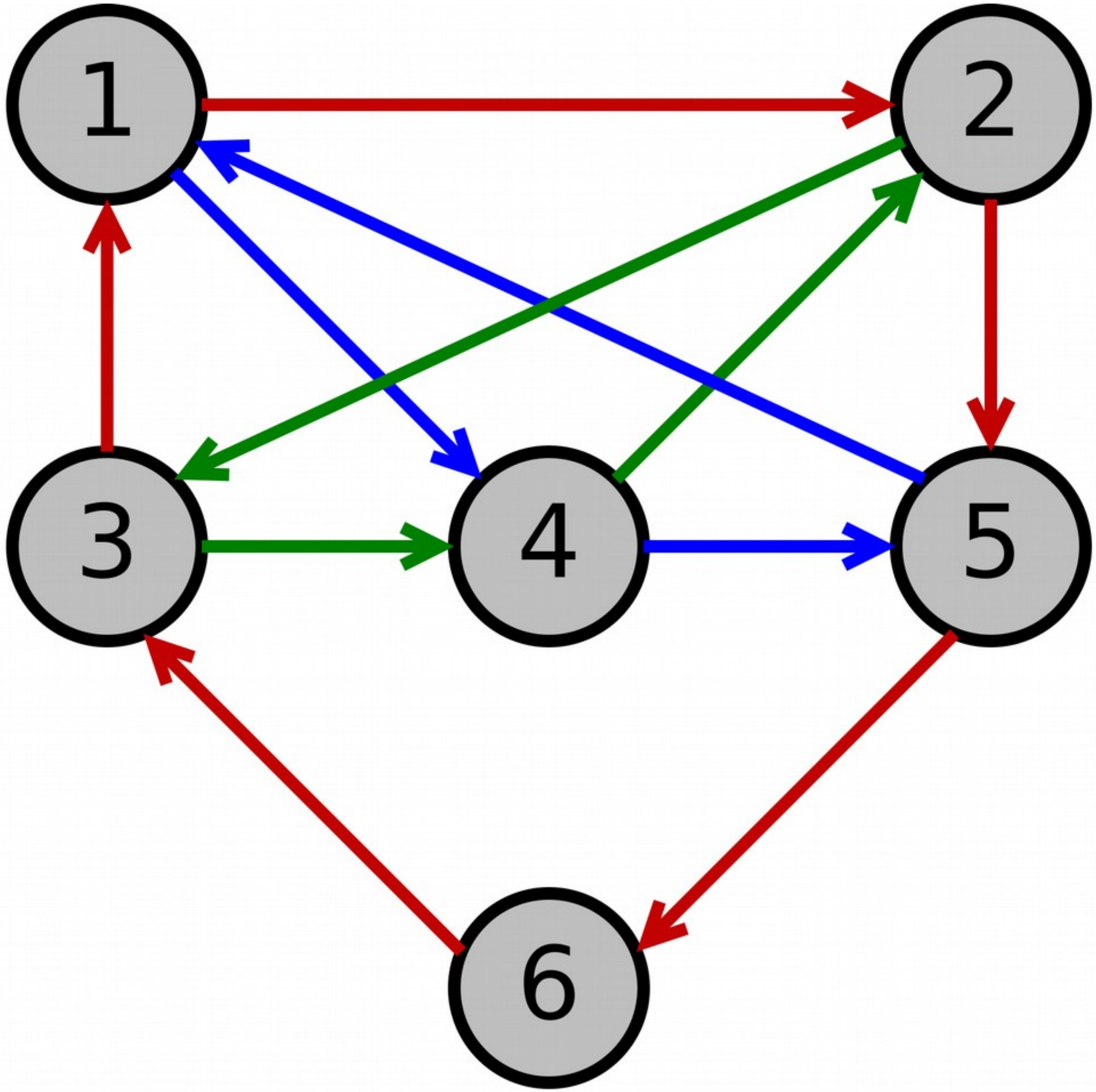


Алгоритм на основе циклов



Алгоритм на основе циклов

6
3
1
2
5
6 1
4
5 2
3
4



Результат: 6 5 4 3 2 4 1 5 2 1 3 6

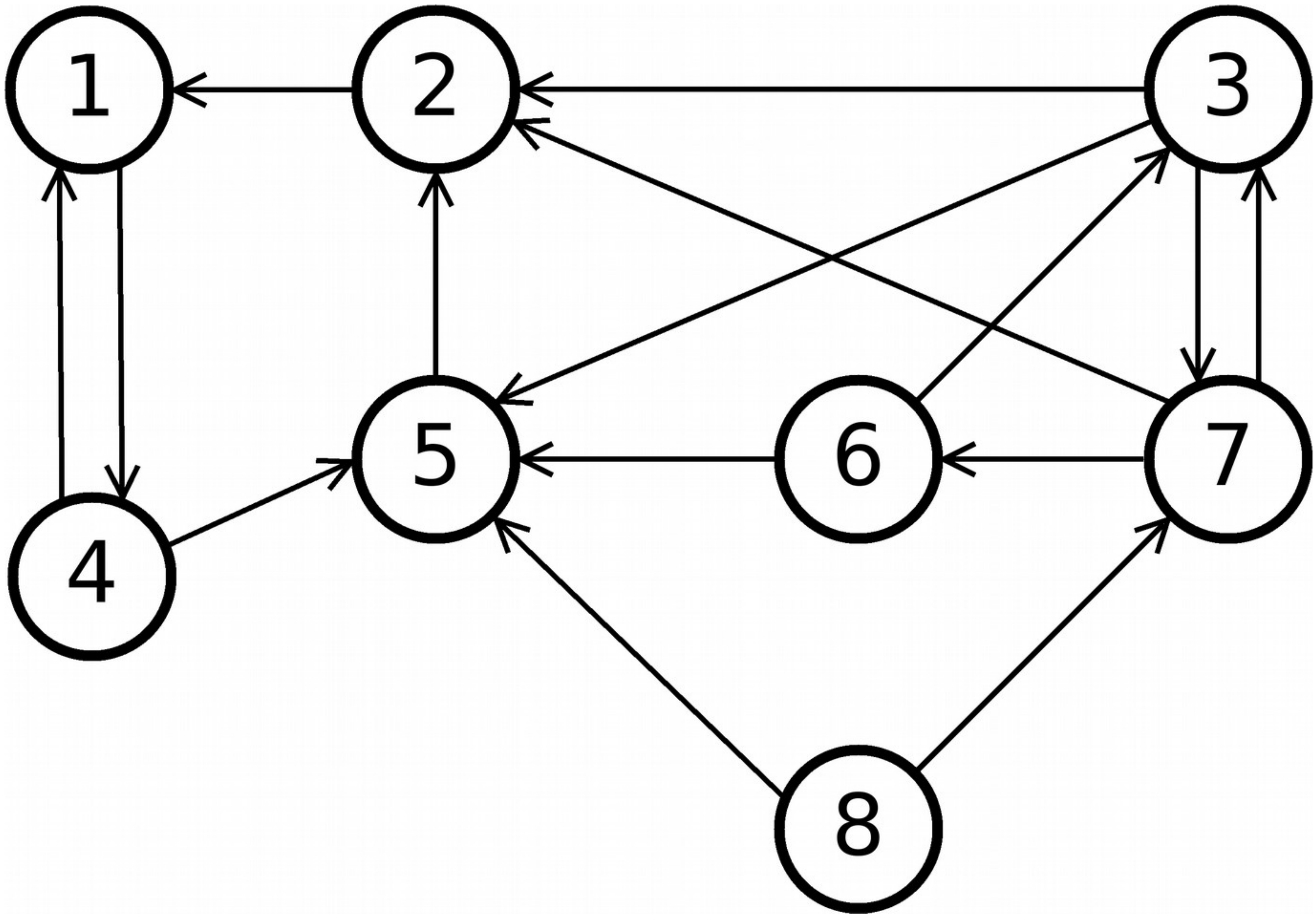
Алгоритм на основе циклов

```
Result — последовательность вершин
s — произвольная стартовая вершина

function FindEulerPath(v)
   для всех (v,v')
        удалить из G ребро (v,v')
        FindEulerPath(v')
   v → Result

FindEulerPath(s)
```

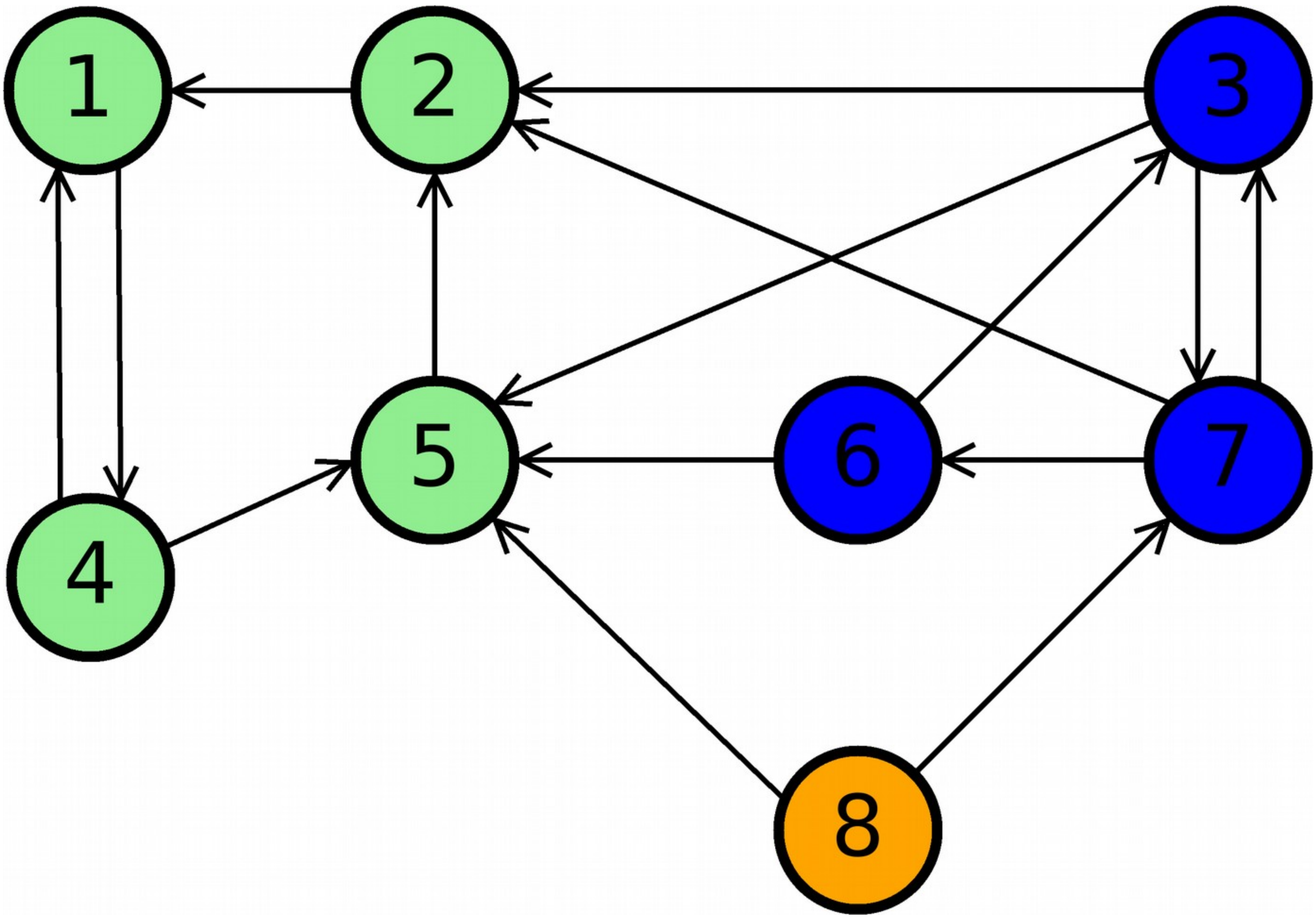
Алгоритм поиска компонент сильной связности

В ориентированном графе G **компонентой сильной связности** называется такое максимальное подмножество вершин, что любые две вершины этого подмножества достижимы друг из друга.

Алгоритм поиска компонент сильной связности



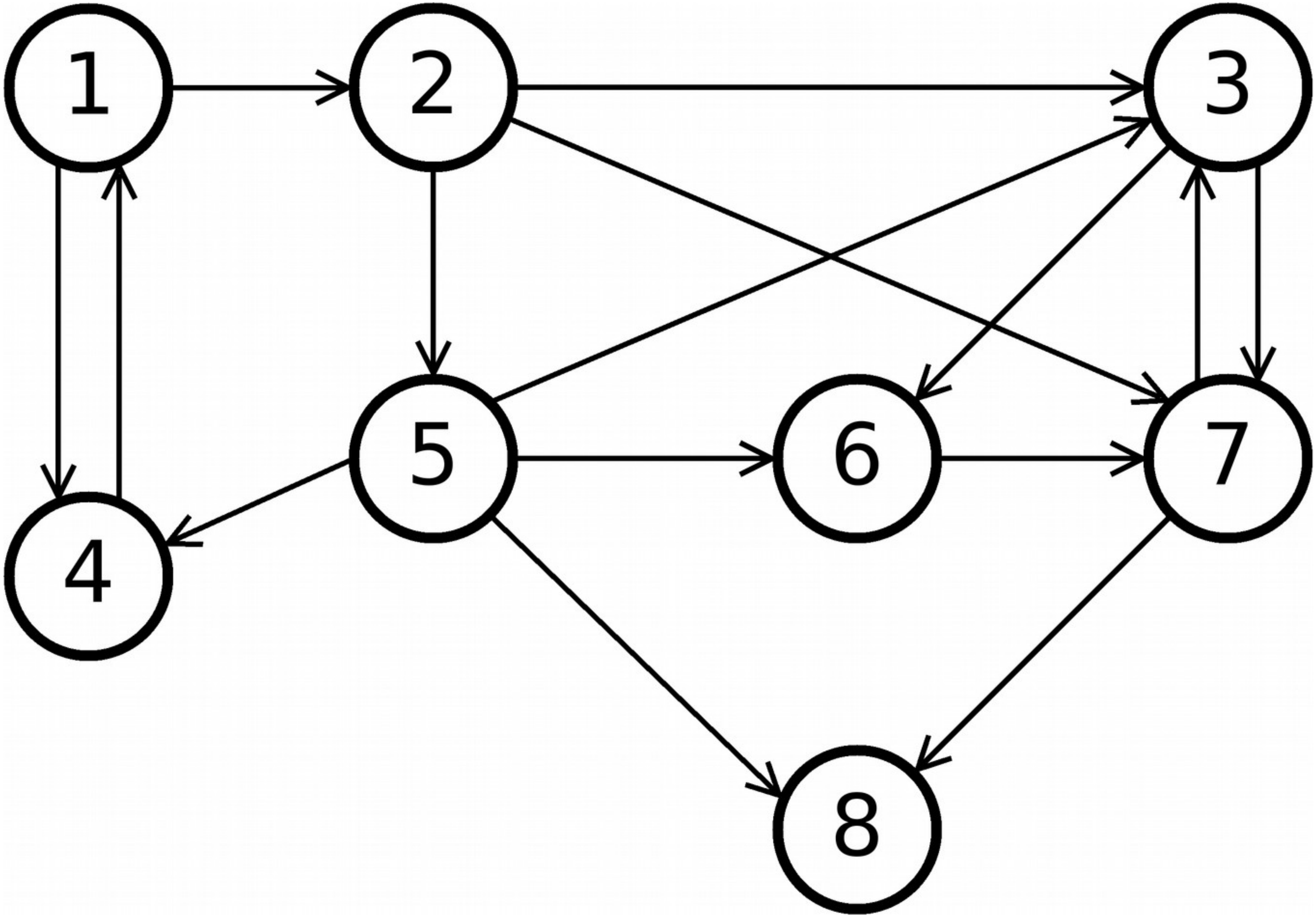
Алгоритм поиска компонент сильной связности



Алгоритм Косарайю (*Kosaraju*)

Идея:
- выполним серию обходов в глубину на графе G^T с инвертированными(обращенными) ребрами;
- запомним *время выхода* из каждой вершины;
- выполним серию обходов в глубину на исходном графе G, начиная с вершины с максимальным временем выхода;
- полученные деревья — компоненты сильной связности.

Алгоритм Косарайю
Обход в глубину на **G^T**

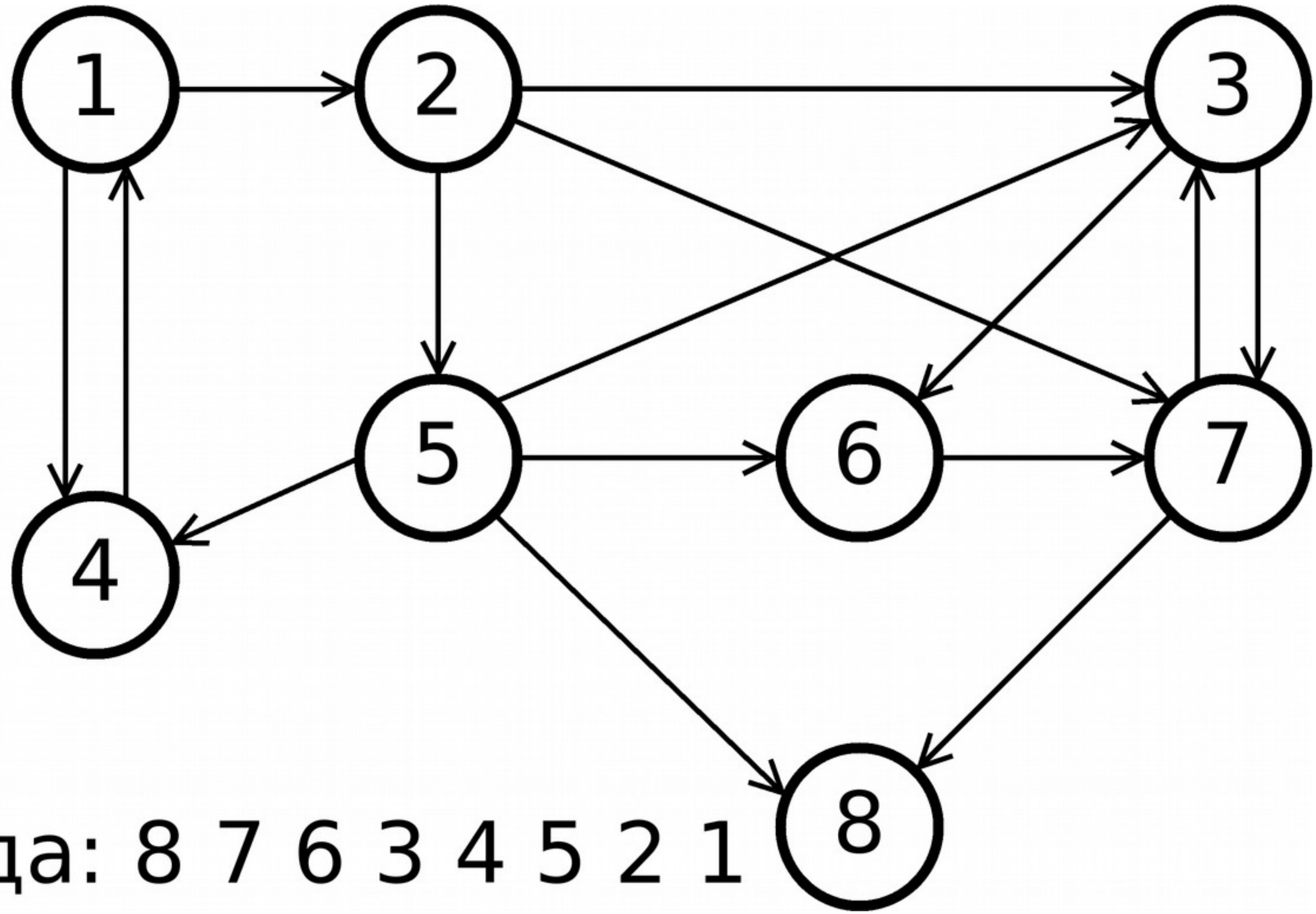


Алгоритм Косарайю
Обход в глубину на G^T

3 1

7 6 4 2

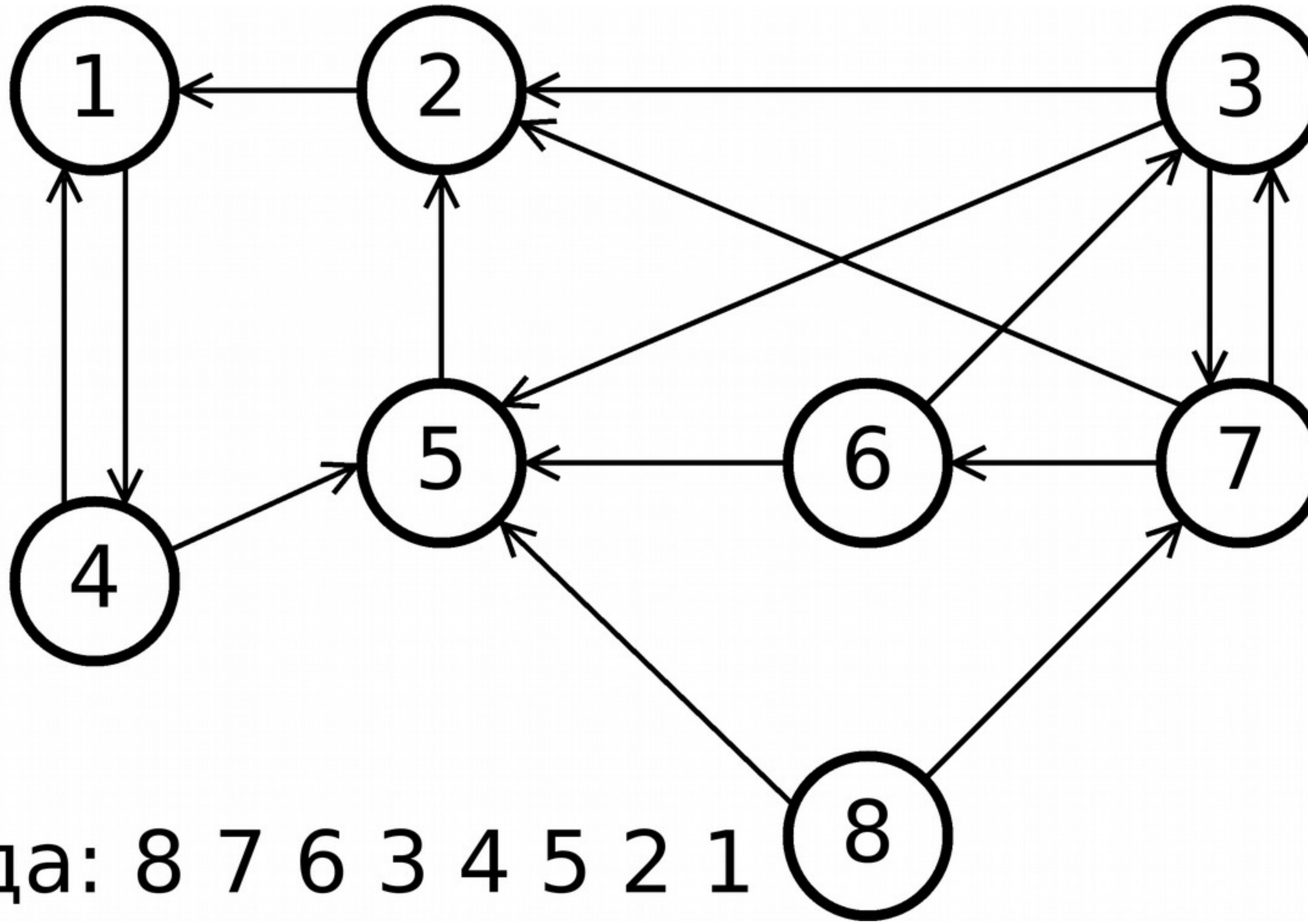
8 5



Время выхода: 8 7 6 3 4 5 2 1

Алгоритм Косарайю
Обход в глубину на **G**

1	3	8
4	7	
5	6	
2		



Время выхода: 8 7 6 3 4 5 2 1

```
Visit[n] — массив посещений
Stack — последовательность вершин

function dfs_inv(x)
   Visit[x] := true
   Для всех y смежных с x
      Если Visit[y] == false
         dfs_inv(y)
   x → Stack

Для G^T
Для всех v: Visit[v] := false
Для всех v из G^T
   Если Visit[v] == false
      dfs_inv(v)

Для G
Для всех v: Visit[v] := false
Пока Stack не пуст
   v ← Stack
   Если Visit[v] == false
      dfs(v)
```