

Алгоритмы на графах

Лекция 4.

Поиск в глубину (завершение).

Бесконтурные графы.

Адигеев Михаил Георгиевич

2022

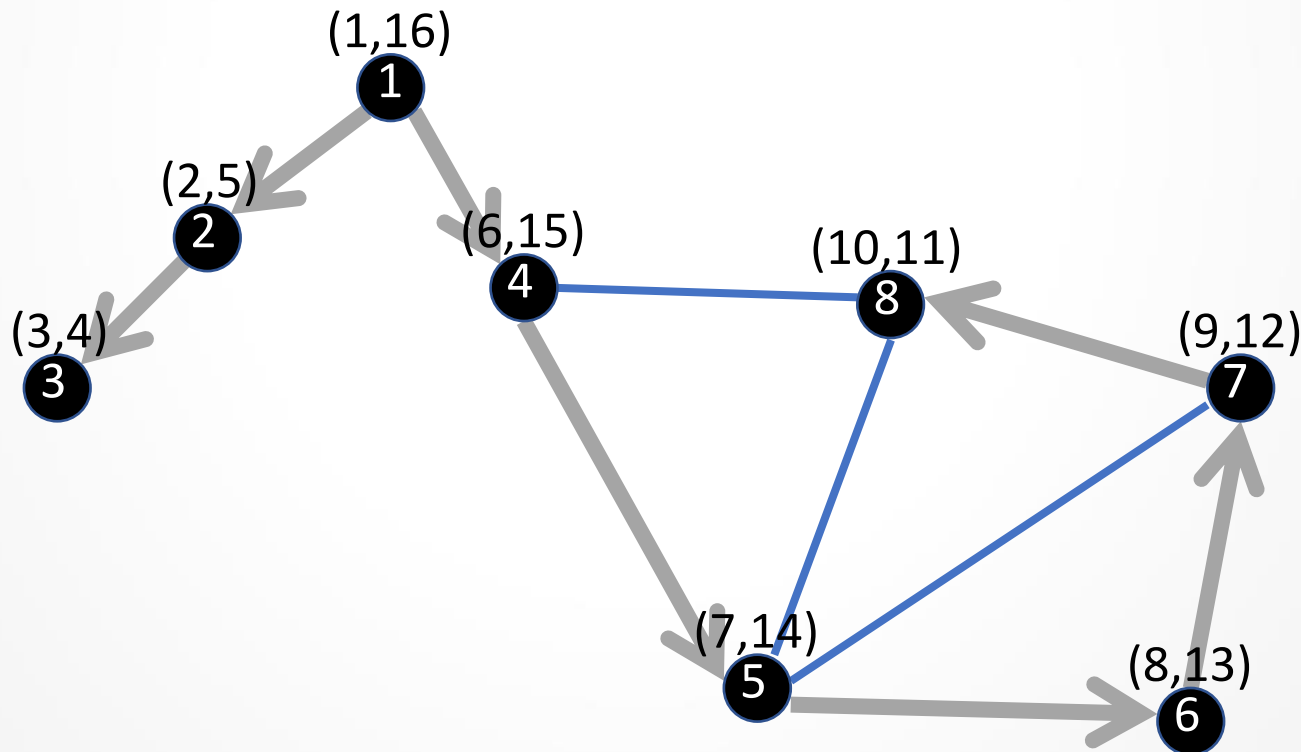
План лекции

1. Обход в глубину неориентированного графа.
 - ✓ Классификация рёбер
 - ✓ Обнаружение мостов
2. Обход в глубину ориентированного графа.
 - ✓ Классификация дуг
 - ✓ Топологическая сортировка вершин
 - ✓ Определение компонент сильной связности

Обход в глубину неориентированных графов

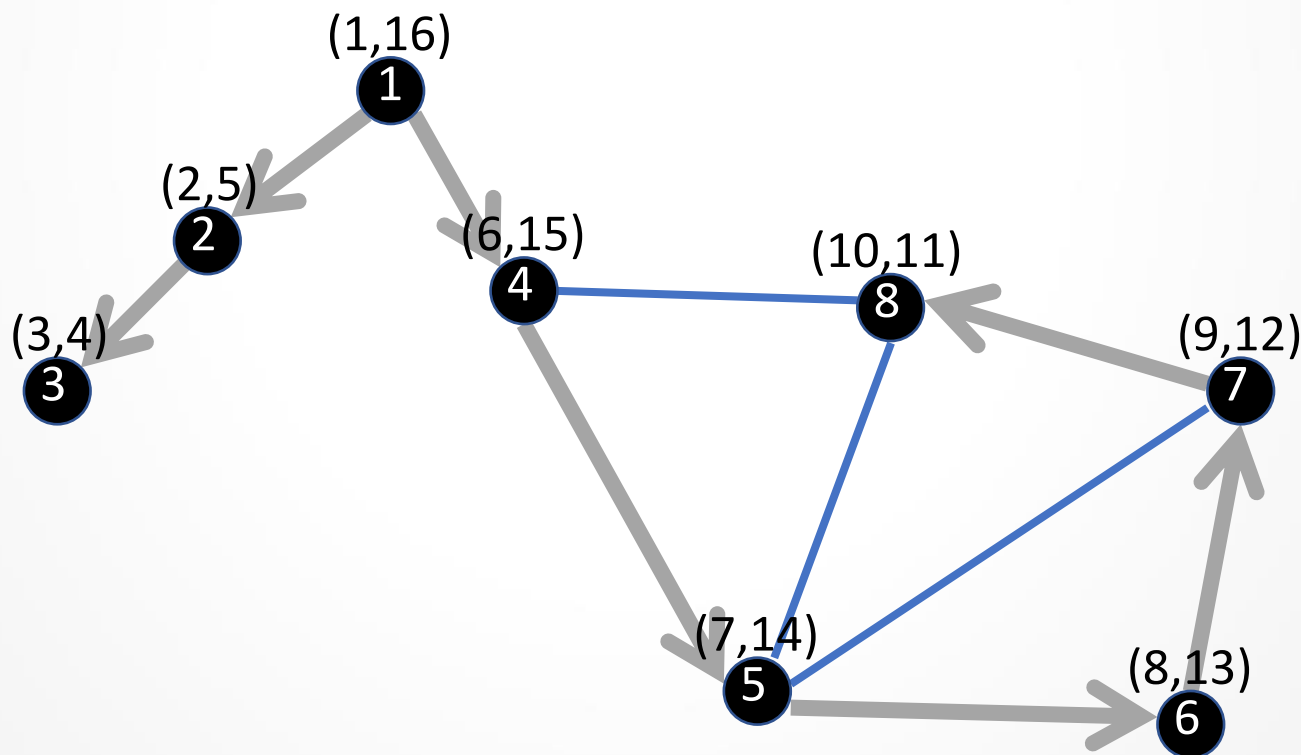
Классификация рёбер

При обходе в глубину неориентированного графа рёбра разделяются на две категории: *древесные* и *обратные*.



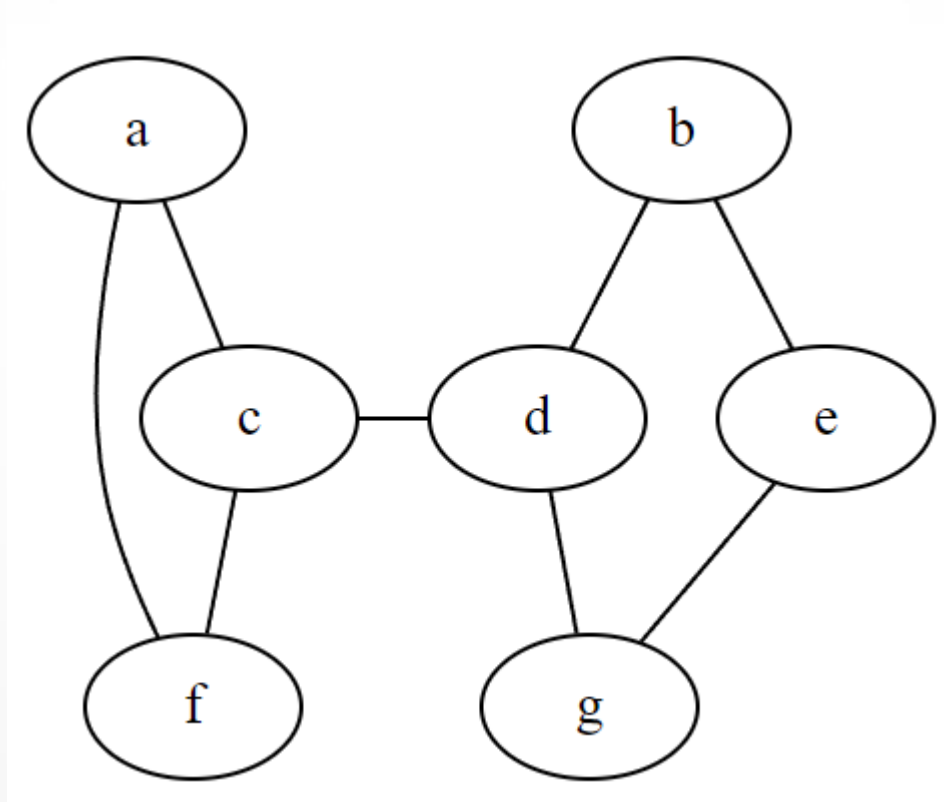
Классификация рёбер

Утверждение. Обратное ребро соединяет вершину с её предшественником (предком в глубинном дереве).



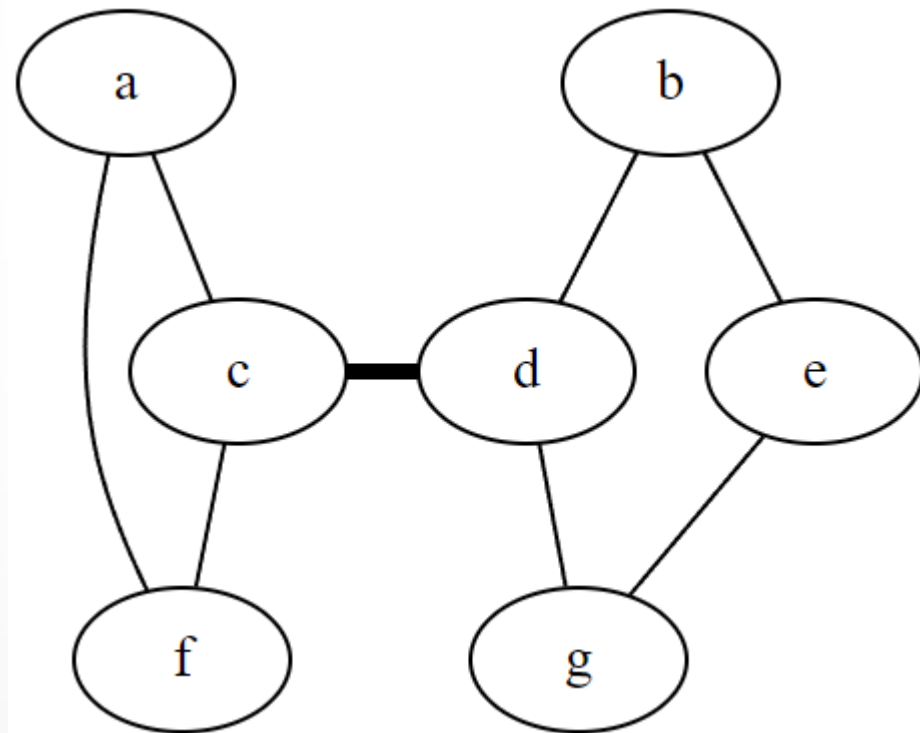
Поиск мостов

Определение. **Мостом** в графе называется ребро, при удалении которого количество компонент связности увеличивается.



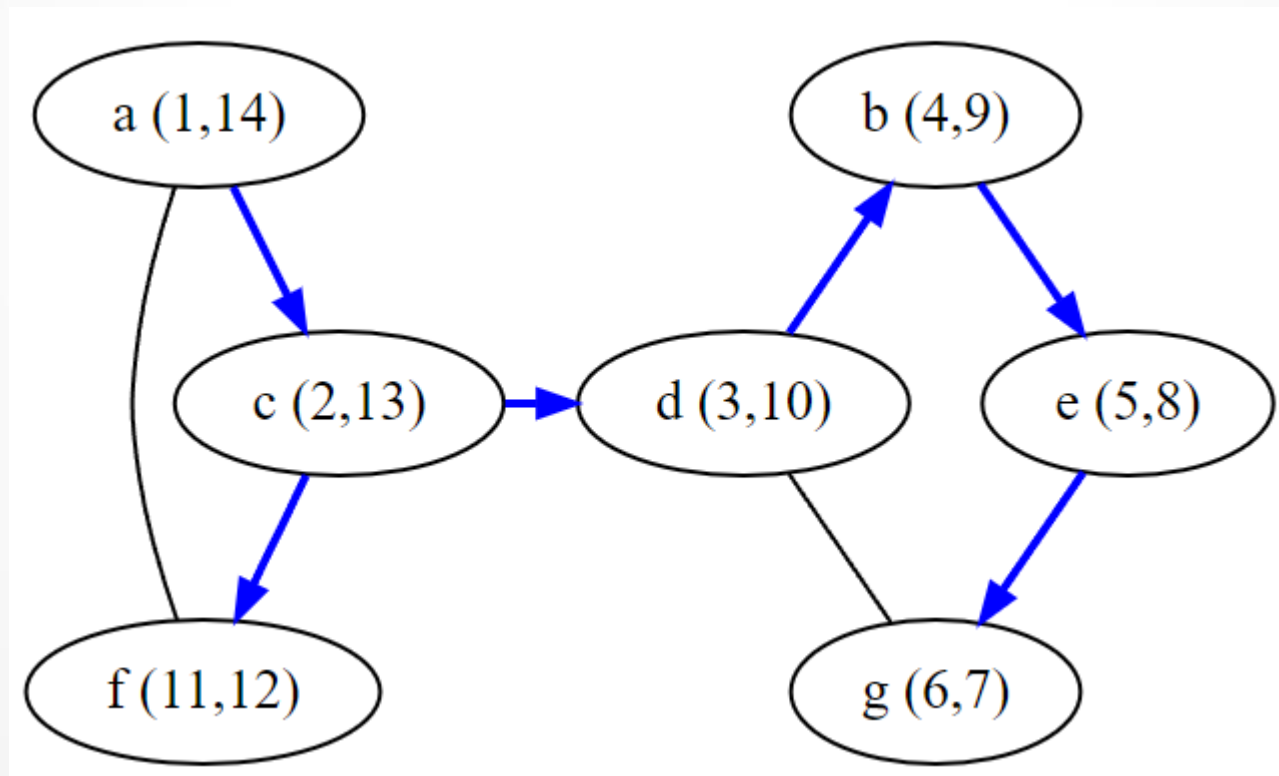
Поиск мостов

Утверждение. Ребро является мостом если и только если оно не входит ни в один цикл.



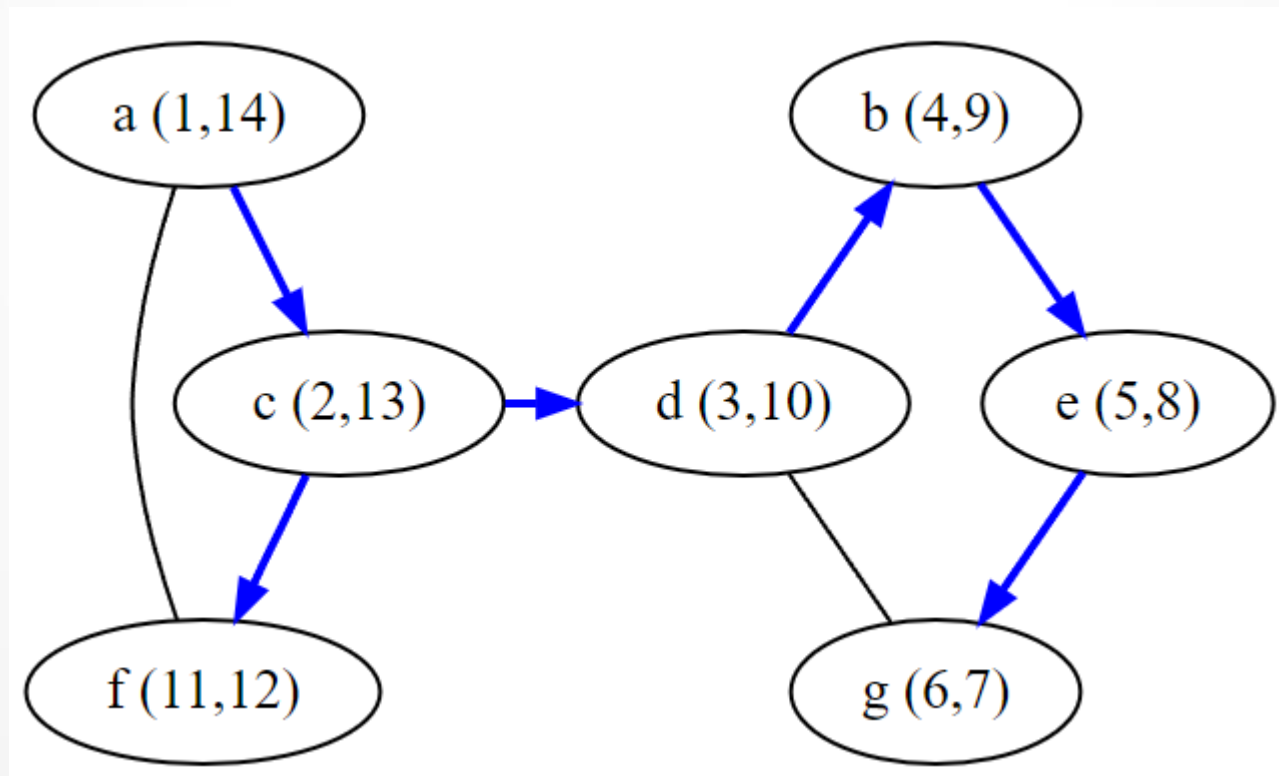
ПОИСК МОСТОВ

Как найти все мосты на графе с помощью поиска в глубину?



Поиск мостов

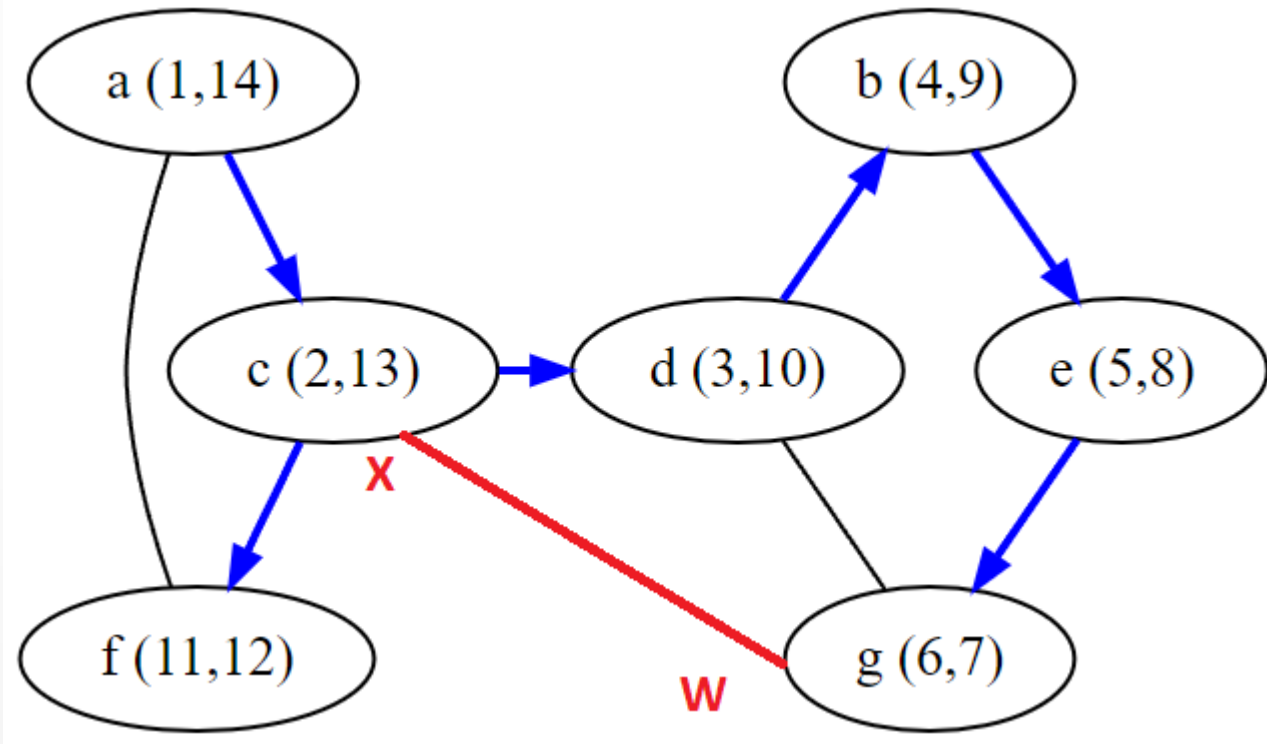
Теорема. Ребро (u, v) является мостом тогда и только тогда, когда это древесное ребро и не существует (обратного) ребра из v или её потомка в u или её предка.



ПОИСК МОСТОВ

Доказательство.

\Leftarrow Пусть (u, v) - древесное ребро и не существует (обратного) ребра из v или её потомка в u или её предка. Удалим это ребро из графа. Предположим, что существует путь, ведущий из v к её предку и (w, x) – последнее ребро на этом пути.

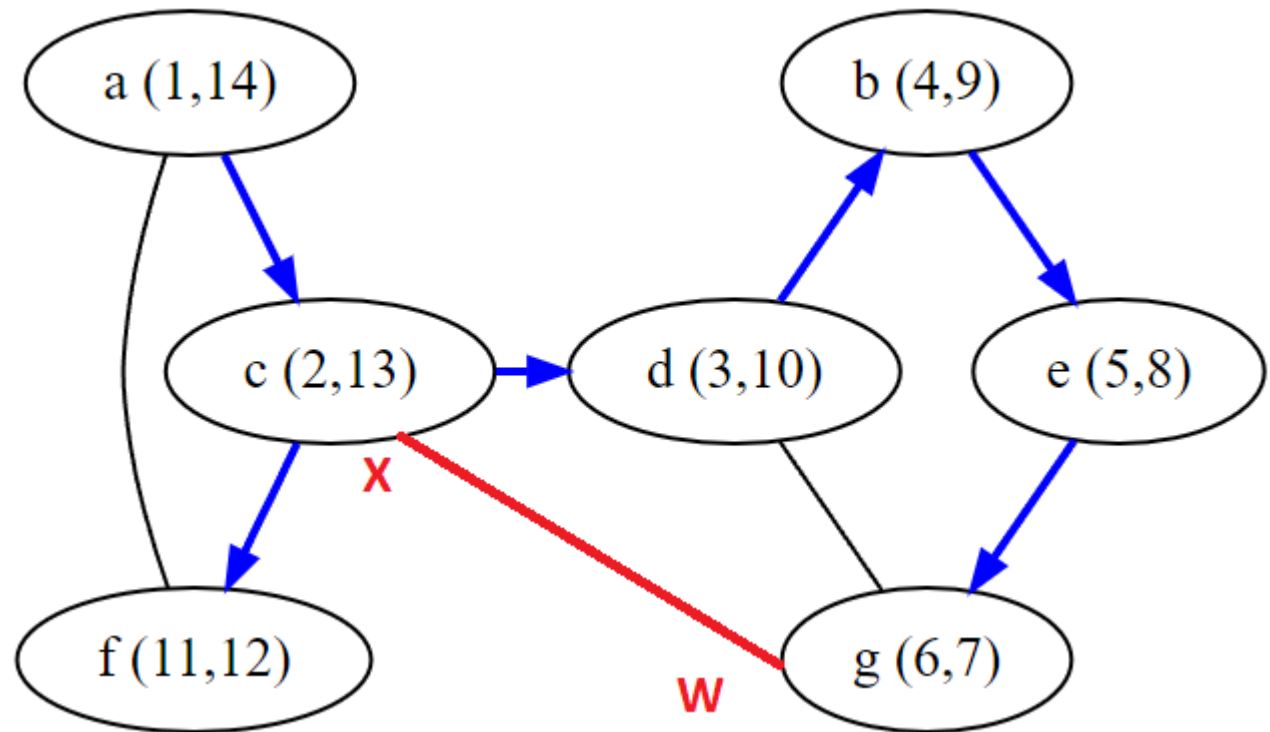


Поиск мостов

(w, x) – не древесное ребро, иначе на глубинном дереве был бы цикл.

(w, x) – не обратное ребро, т.к. это противоречило бы условию теоремы.

Получили противоречие.



Поиск мостов

Как проверять условие теоремы («не существует (обратного) ребра из v или её потомка в u или её предка»)?

При обходе в глубину будем рассчитывать ещё один показатель $Low[]$.

По определению, $Low[v]$ = минимальное время входа ($Time_In$) для вершин, достижимых из v или её потомков.

Более точно: $Low[v]$ = минимум следующих величин:

- $Time_In[v]$;
- $Low[x]$ для всех x – потомков v ;
- $Time_In[w]$ для всех обратных рёбер (x,w) , где x – (нестрогий) потомок v , w – предок v .

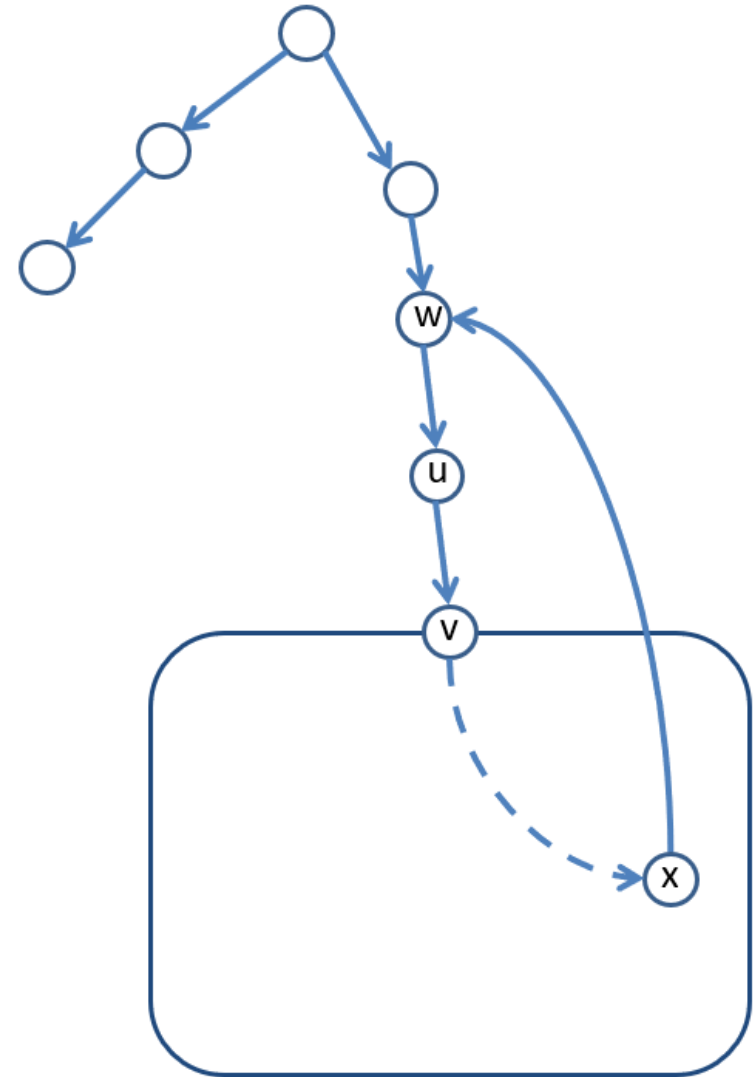
Поиск мостов

$Low[v]$ = минимум следующих величин:

- $Time_In[v]$;
- $Low[x]$ для всех x – потомков v ;
- $Time_In[w]$ для всех обратных рёбер (x,w) , где x – (нестрогий) потомок v , w – предок v .

Тогда: (u,v) – мост, если и только если $Low[v] > Time_in[u]$.

Сложность поиска мостов: $O(n + m)$.



Алгоритм поиска мостов

DFS (G)

For each $v \in V$:

 State[v] := 'unvisited';

 Pred[v] := NULL;

 Time_In[v] := NULL;

 Time_Out[v] := NULL;

 Low[v] := NULL;

CurTime := 0;

For each $v \in V$:

 If State[v] = 'unvisited'

 DFS_Visit(v);

Алгоритм поиска мостов

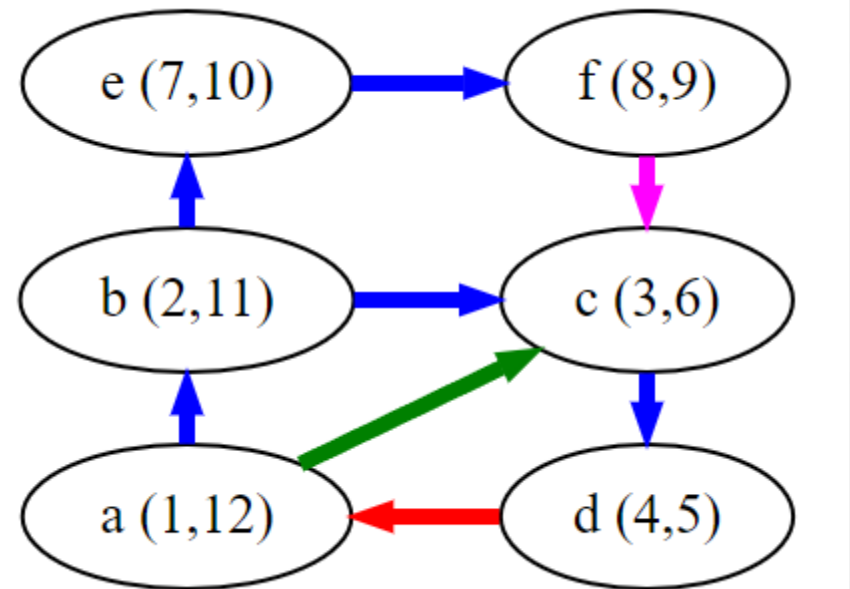
```
DFS_Visit(v)
State[v] := 'visited';
CurTime := CurTime + 1;
Time_In[v] := CurTime;
Low[v] := Time_In[v];
For each u in Adj(v)
    If State[u] = 'unvisited'
        Pred[u] := v;
        DFS_Visit(u);
        Low[v] := min(Low[u], Low[v]);
    Else
        Low[v] := min(Low[v], Time_In[u]);
    If Low[v] > Time_In[u] then (v,u) - мост;
State[v] := 'processed';
CurTime := CurTime + 1;
Time_Out[v] := CurTime;
```

Обход в глубину ориентированных графов

Классификация дуг

При обходе в глубину ориентированного графа дуги разделяются на четыре категории:

- 1) *Древесные* – по ним проходим при обходе
- 2) *Прямые* – ведут из древесного предка в потомка
- 3) *Обратные* – ведут из древесного потомка в предка
- 4) *Поперечные* – ведут в вершину, не являющуюся предком или потомком.



Классификация дуг

Как определить тип дуги (u, v) при обходе в глубину?

Используем пометки $Pred$, $Time_In$, $Time_Out$:

1) *Древесная*: $u = Pred[v]$

2) *Прямая*:

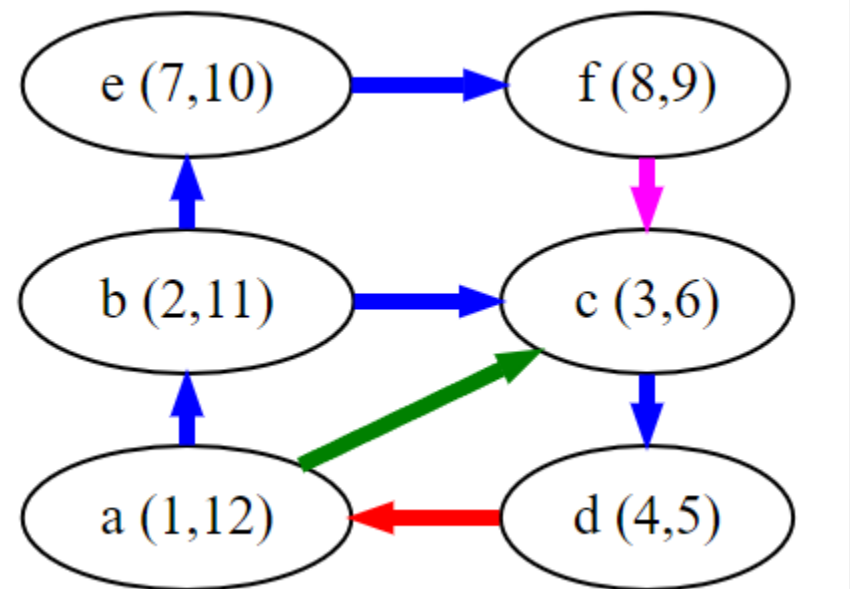
$Time_In[u] < Time_In[v]$
 $Time_Out[v] < Time_Out[u]$

3) *Обратная*

$Time_In[u] > Time_In[v]$
 $Time_Out[v] > Time_Out[u]$

4) *Поперечная*:

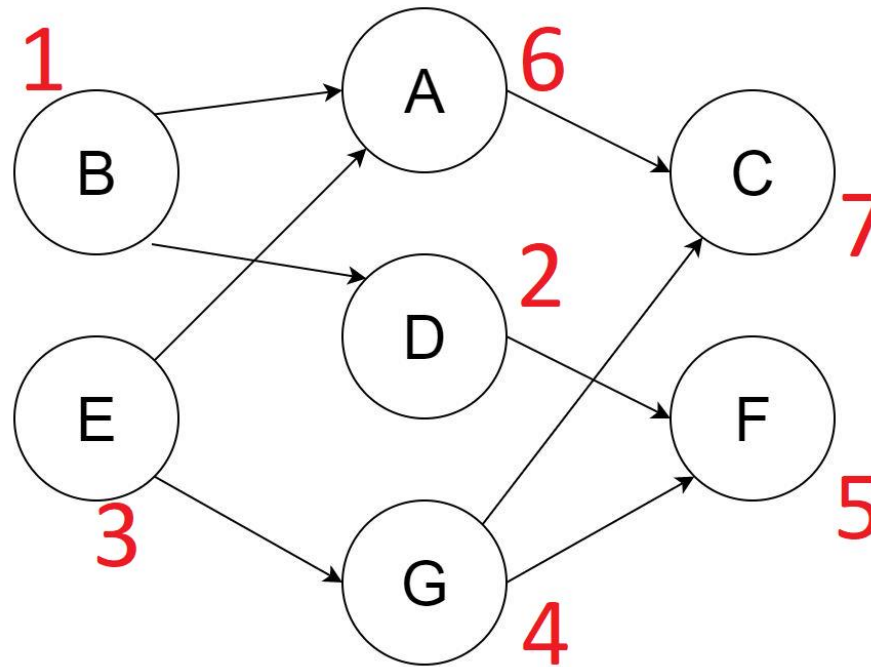
$Time_Out[v] < Time_In[u]$



Топологическая сортировка

Пусть $G(V,E)$ – ориентированный граф.

Топологической сортировкой (правильной нумерацией) называется нумерация вершин $\tau: V \rightarrow \{1, \dots, |V|\}$, при которой каждая дуга ведёт из вершины с меньшим номером в вершину с большим номером: $\forall (u, v) \in E \tau(u) < \tau(v)$.

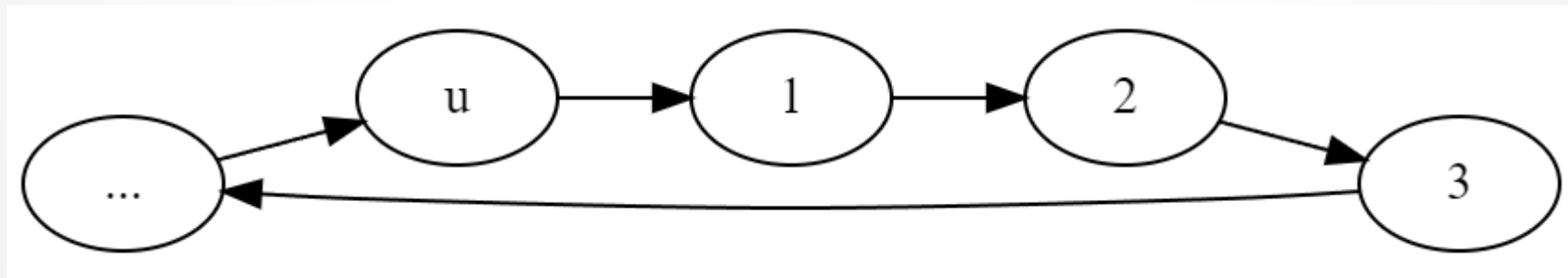


Топологическая сортировка

Теорема. Для графа G существует топологическая сортировка \Leftrightarrow граф является *бесконтурным*.

Доказательство.

\Rightarrow От противного: пусть граф не является бесконтурным. Тогда вершины, лежащие на контуре, невозможно пронумеровать правильным образом.



Топологическая сортировка

⇐ Пусть $G(V,E)$ – бесконтурный граф. Покажем, как для него можно построить топологическую сортировку.

Утверждение. В каждом бесконтурном графе есть как минимум один источник и как минимум один сток.

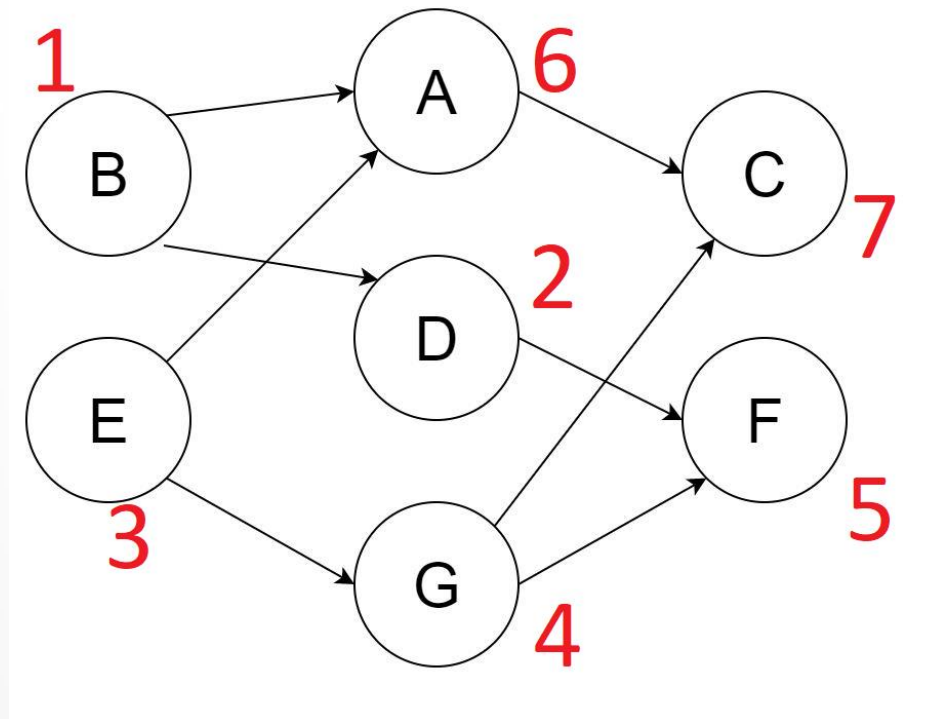
Алгоритм топологической сортировки:

1. Создаём счётчик, инициализируем значением 1.
2. Пока $|V| > 0$
 - Находим источник и присваиваем ему очередной номер.
 - Удаляем этот источник из графа.

Топологическая сортировка

Полученная нумерация является топологической сортировкой.

- 1) Все вершины пронумерованы (т.к. после удаления каждой вершины граф остаётся бесконтурным).
- 2) Все дуги ведут из вершин с меньшим номером в вершину с большим номером.

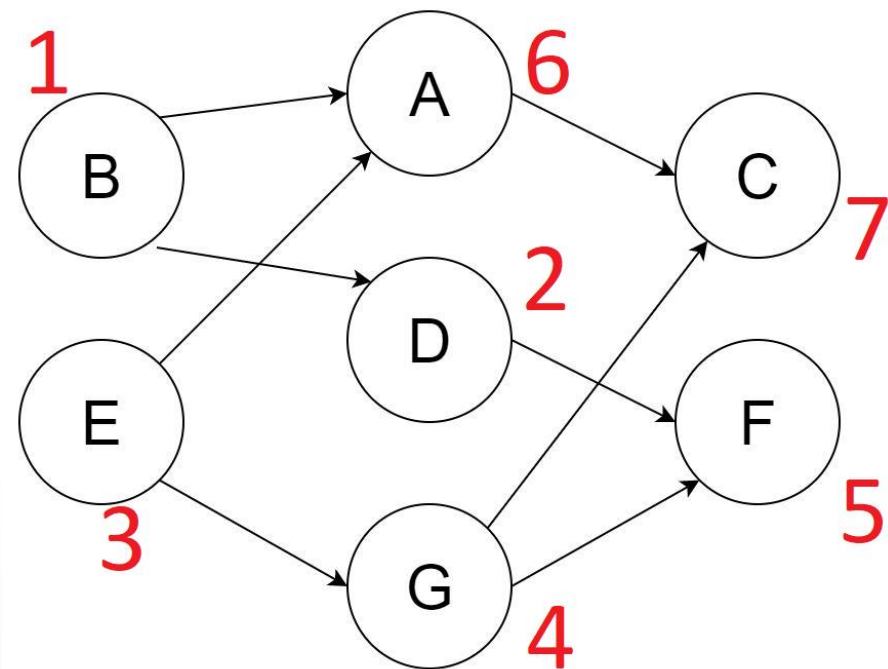


Топологическая сортировка

Для построения топологической сортировки можно использовать и обход графа в глубину.

1. Заводим счётчик «топологических номеров». Инициализируем количеством вершин ($n = |V|$).
2. Обходим граф в глубину. Перед выходом из вершины присваиваем ей очередной «топологический номер», счётчик уменьшаем на 1.

Сложность построения топологической сортировки: $O(n + m)$.



Топологическая сортировка

DFS TopSort (G)

For each $v \in V$:

 State[v] := 'unvisited';

 Pred[v] := NULL;

 Time_In[v] := NULL;

 Time_Out[v] := NULL;

 TopNum[v] := NULL;

CurTime := 0;

CurTopNum := n;

For each $v \in V$:

 If State[v] = 'unvisited'

 DFS_TopSort_Visit(v);

Топологическая сортировка

```
DFS TopSort Visit(v)
State[v] := 'visited';
CurTime := CurTime + 1;
Time_In[v] := CurTime;
For each u in Adj(v)
    If State[u] = 'unvisited'
        Pred[u] := v;
        DFS_Visit(u);
State[v] := 'processed';
CurTime := CurTime + 1;
Time_Out[v] := CurTime;
TopNum[v] := CurTopNum;
CurTopNum := CurTopNum - 1;
```

Практические задачи

Практические задачи

Задача «Алфавит Нитал»

В языке народа Нитал используются буквы латинского алфавита. До наших дней дошел документ, в котором есть глоссарий. Очевидно, что слова в нем расположены по алфавиту, но никто не знает порядка букв в алфавите, использовавшемся народом Нитал.

Требуется написать программу, которая по глоссарию сможет восстановить порядок букв в алфавите народа Нитал.

Практические задачи

Формат входных данных

В первой строке число N - количество слов в глоссарии. В каждой следующей одно слово.

Формат выходных данных

Одна строка, содержащая буквы языка Нитал в алфавитном порядке и без пробелов. В этой строке должны содержаться все буквы, которые входили в слова глоссария.

Практические задачи

Задача «Алфавит Нитал»

Пример

input.txt	output.txt
3	WXY
WXX	
WYX	
XW	