Алгоритмы на графах

Лекция 4.

Поиск в глубину (завершение).

Эйлеровы циклы.

Адигеев Михаил Георгиевич

2024

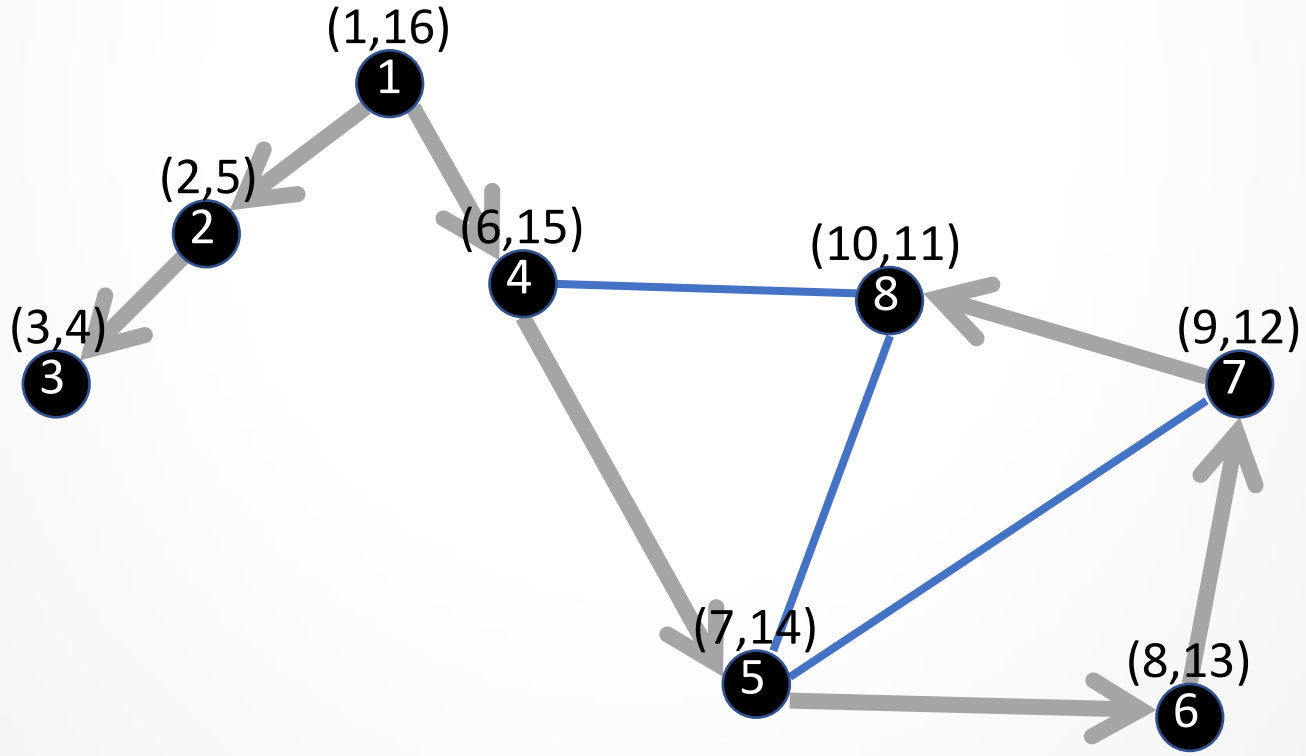
План лекции

1. Обход в глубину неориентированного графа.
   - ✓ Классификация рёбер.
   - ✓ Обнаружение мостов.
2. Эйлеровы циклы.
   - ✓ Определение и критерий эйлеровости и полуэйлеровости.
   - ✓ Нахождение эйлеровых циклов.
     - Алгоритм Флёри.
     - Алгоритм Хирхольцера.

Обход в глубину неориентированных графов

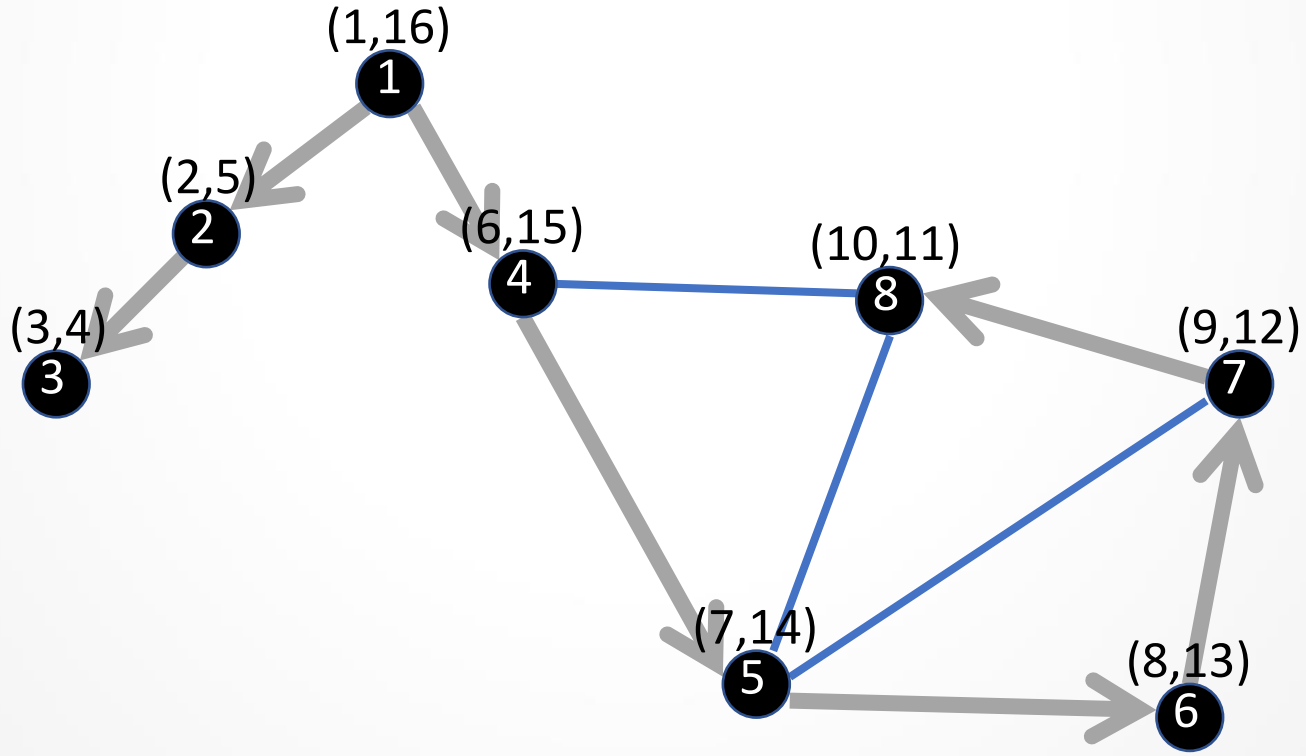
Классификация рёбер

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



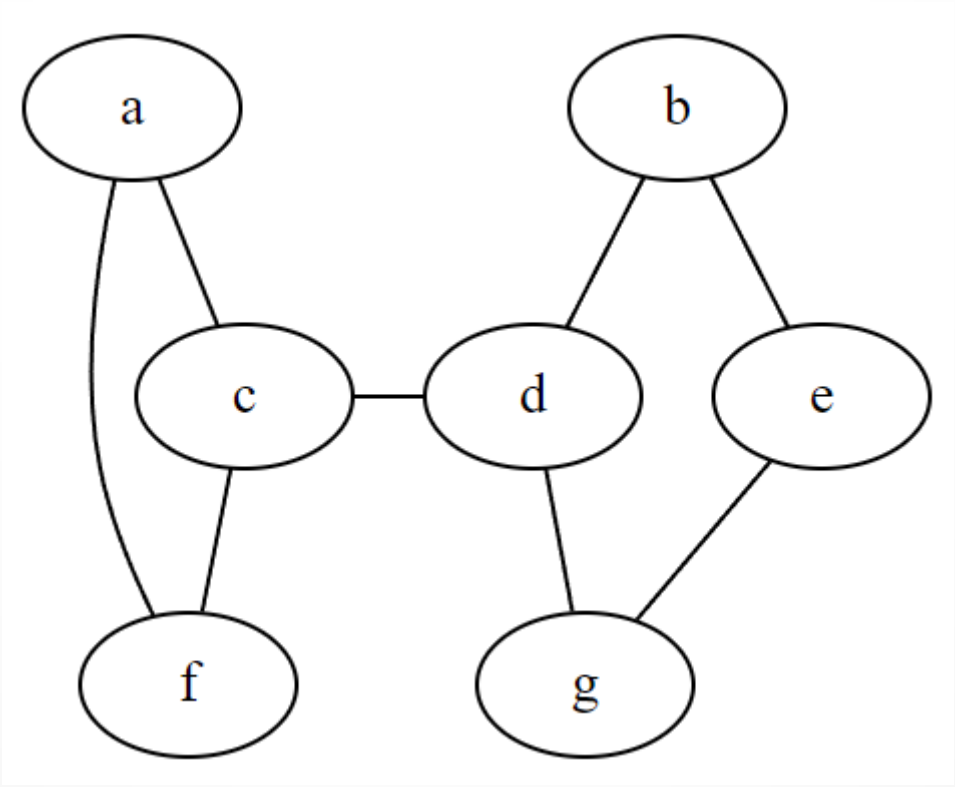
Классификация рёбер

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



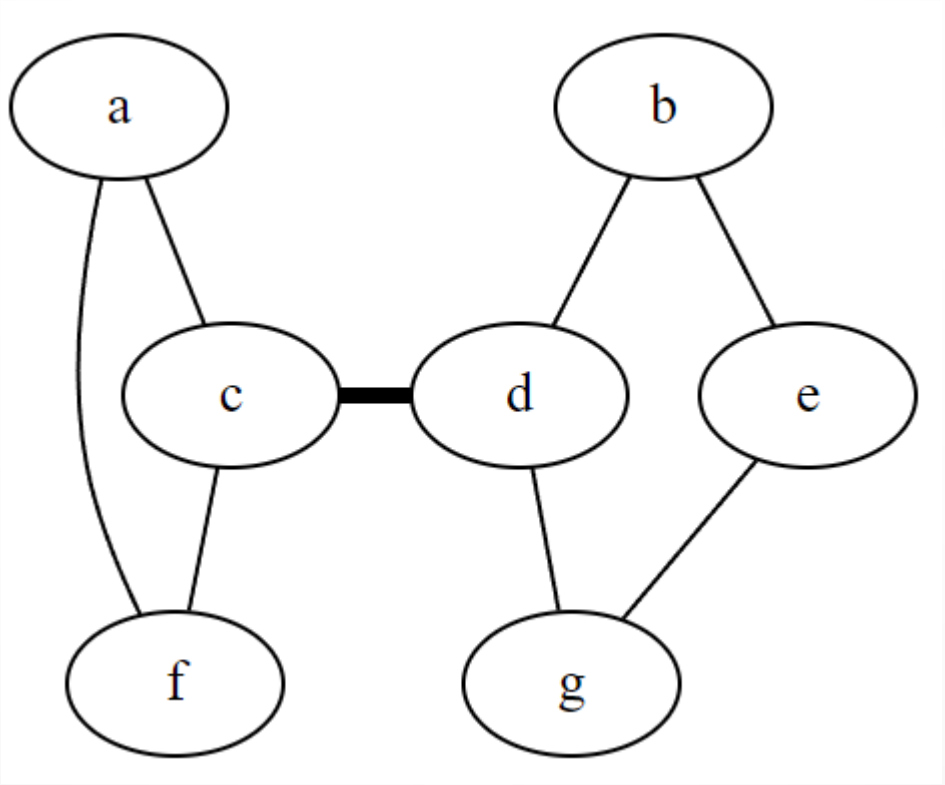
Поиск мостов

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



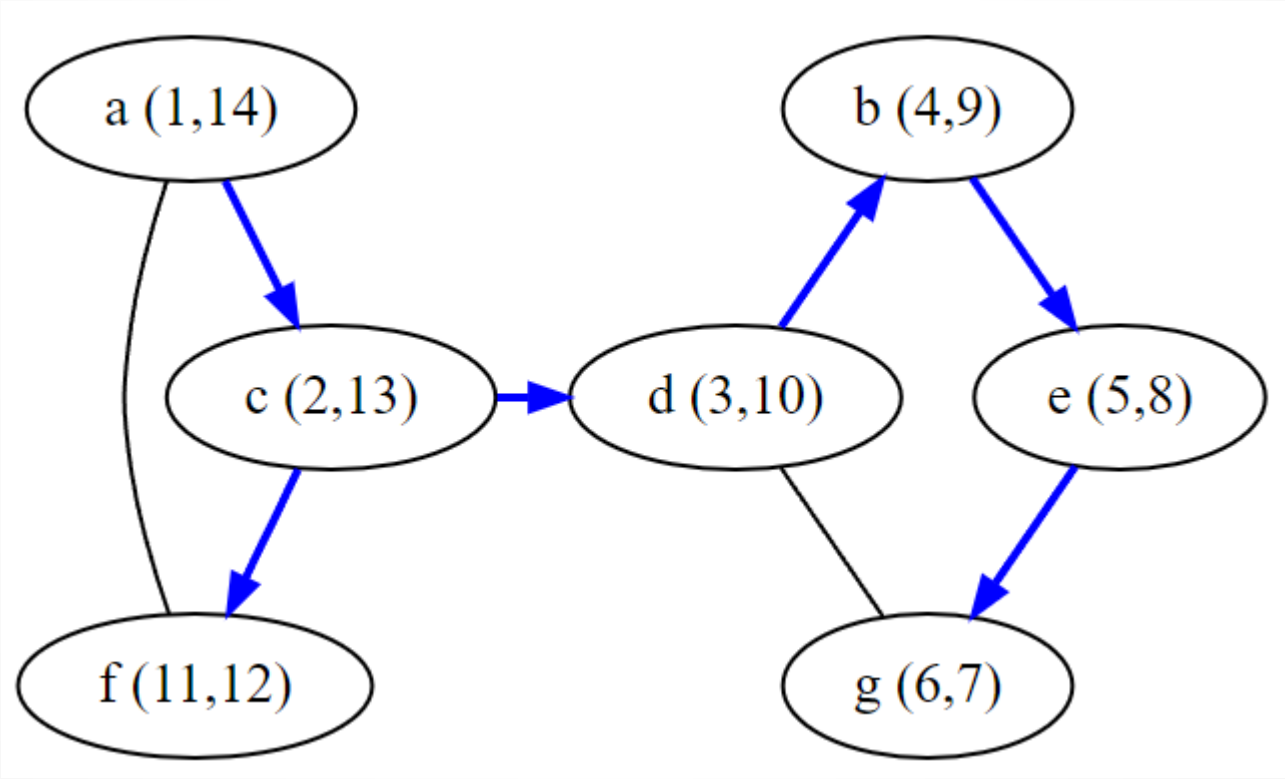
Поиск мостов

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



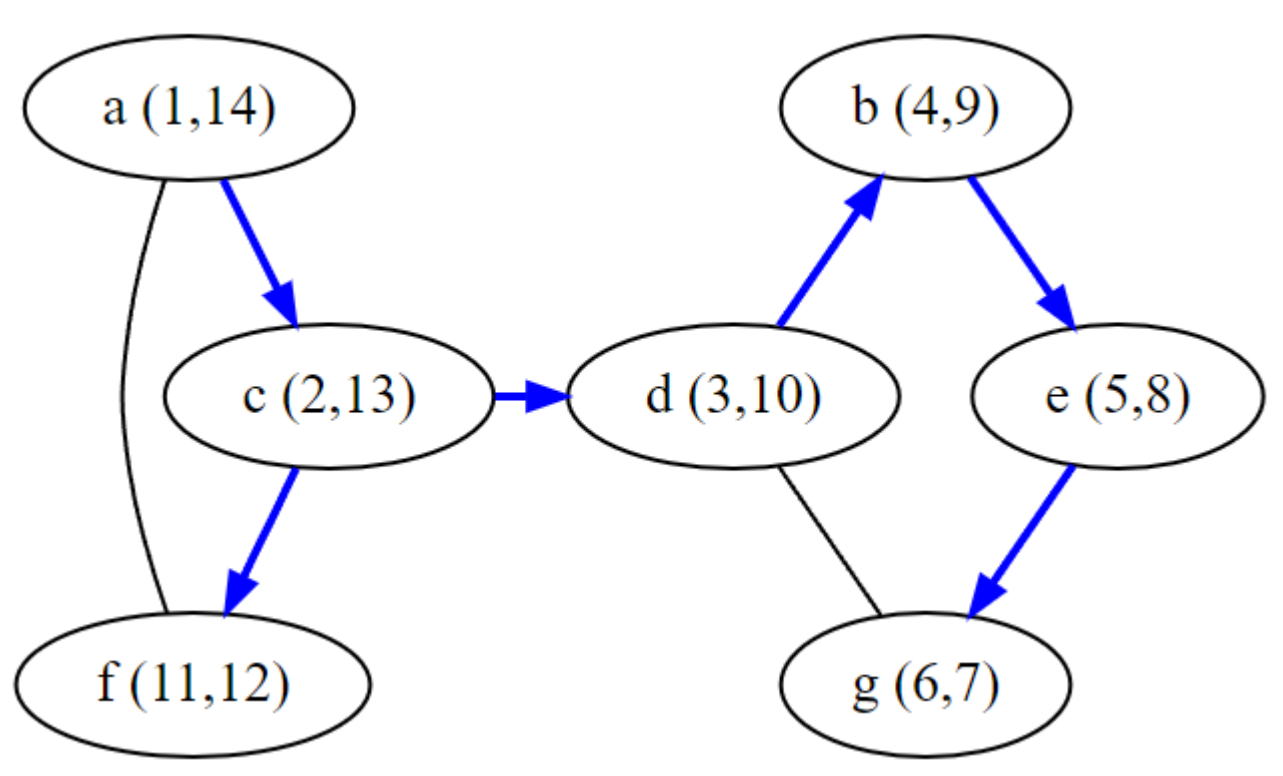
Поиск мостов

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



Поиск мостов

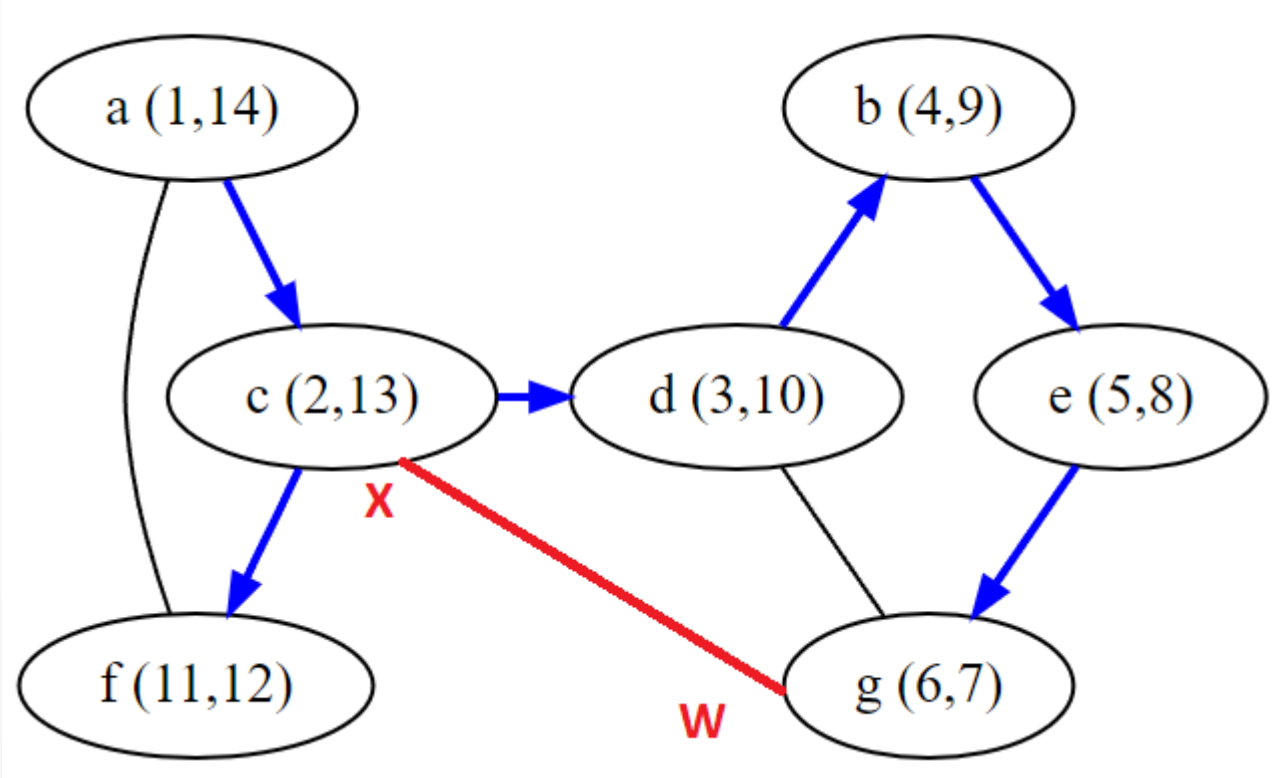
Теорема. Ребро `(u,v)` является мостом тогда и только тогда, когда это древесное ребро и не существует (обратного) ребра из `v` или её потомка в `u` или её предка.



Поиск мостов

Доказательство.

⇐ 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.

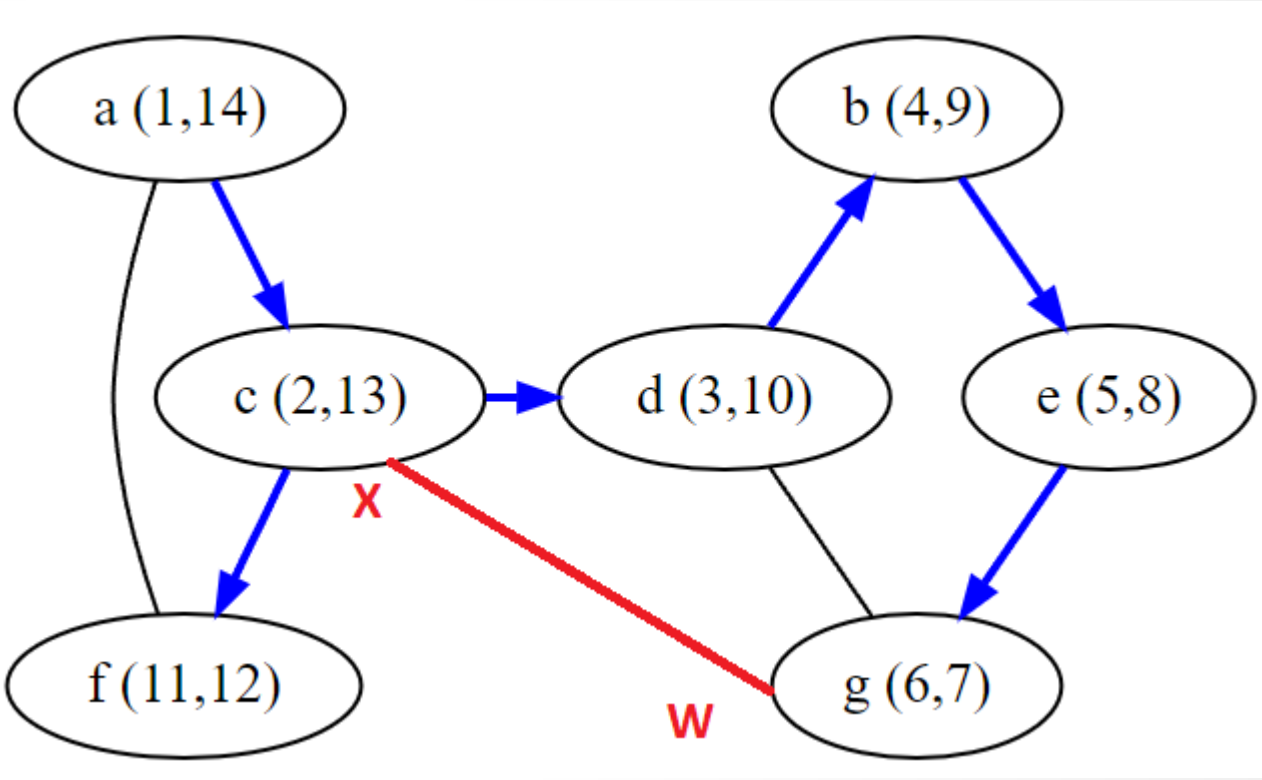


Поиск мостов

`(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.

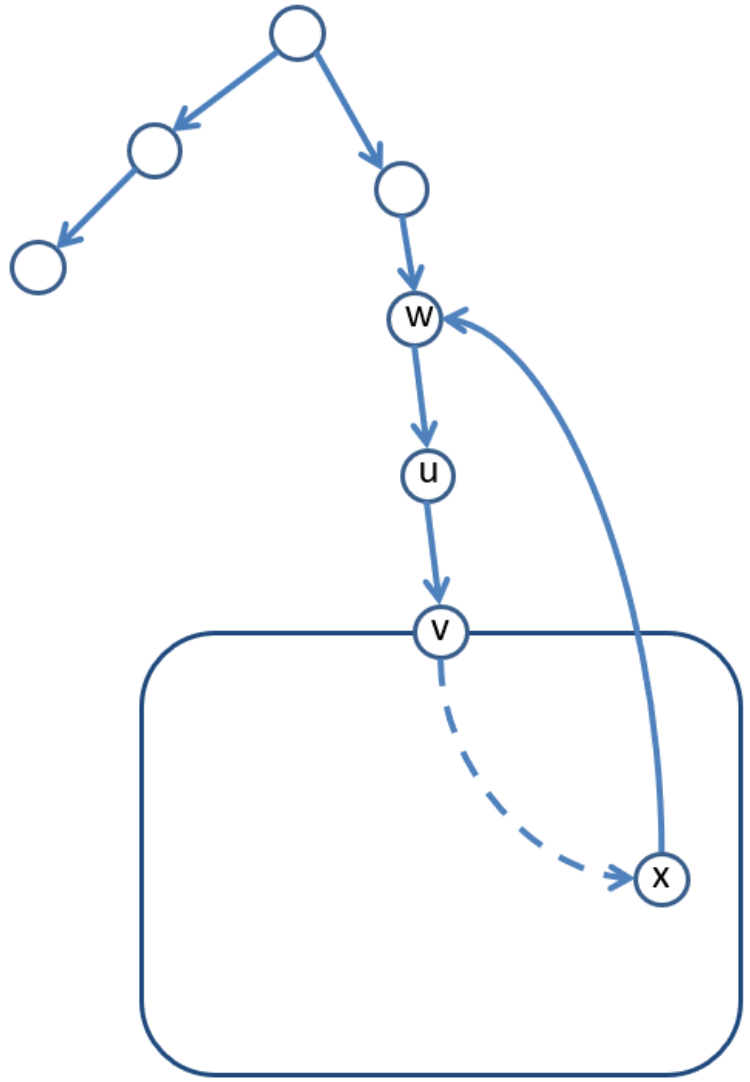
Более точно: `Low[v]` = минимум следующих величин:

- `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 ∈ V:
   State[v] := 'unvisited';
   Pred[v] := NULL;
   Time_In[v] := NULL;
   Time_Out[v] := NULL;
   Low[v] := NULL;
CurTime : = 0;
For each v ∈ V:
   If State[v] = 'unvisited'
        DFS_Visit(v);
```

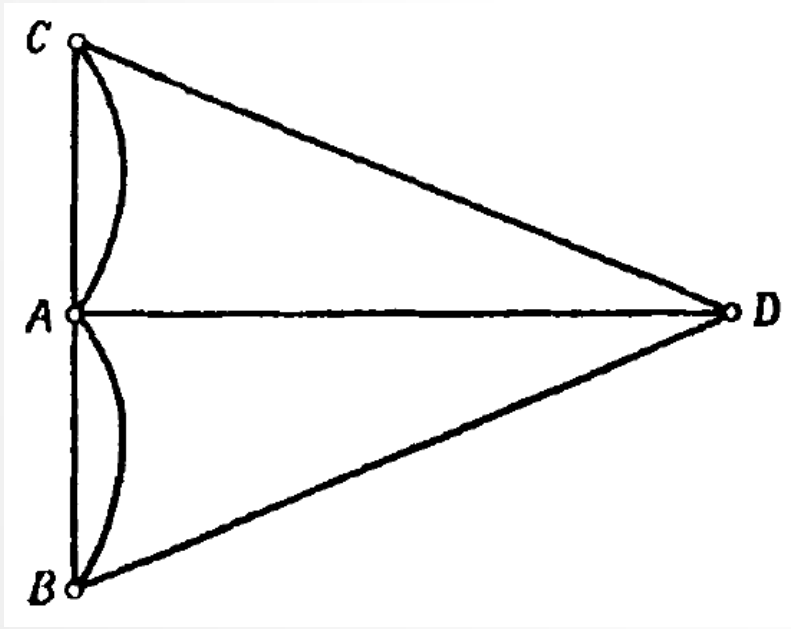
Алгоритм поиска мостов

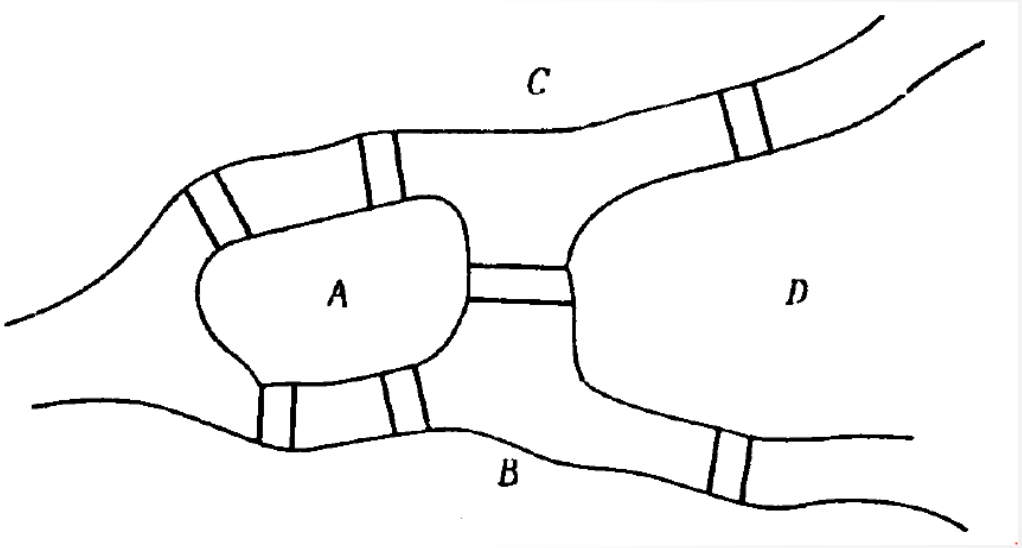
```
DFS_Visit(v)
State[v] := 'visited';
CurTime := CurTime + 1;
Time_In[v] := CurTime;
Low[v] := Time_In[v];
For each u in Adj(v)
    If State[u] = 'unvisited'
          Pred[u] := v;
          DFS_Visit(u);
          Low[v] := min(Low[u], Low[v]);
    Else
          Low[v] := min(Low[v], Time_In[u]);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

Эйлеровы циклы

Эйлеровость

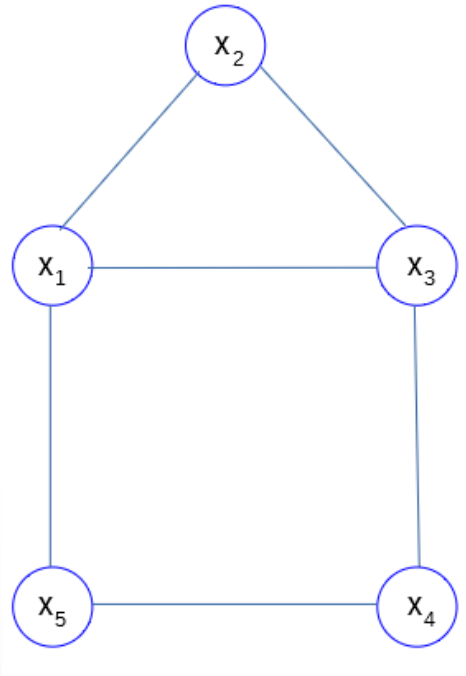
Первая задача теории графов (1736 г.) - задача о Кёнигсбергских мостах. Решена Л.Эйлером.

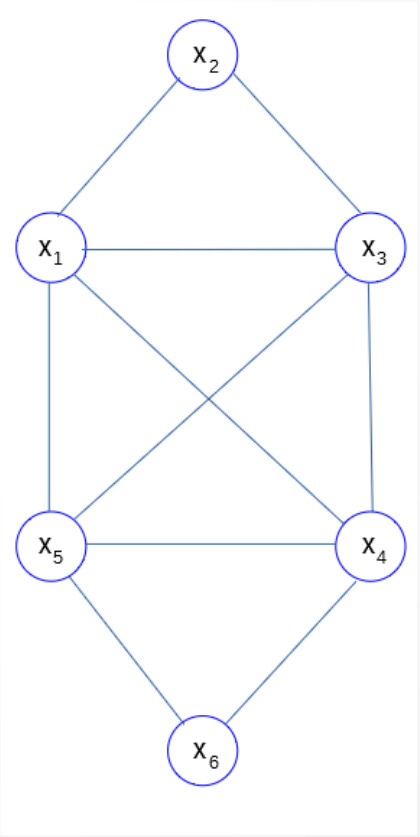




Эйлеровость

Определение. Связный граф называется *эйлеровым*, если на нём существует простой *цикл*, проходящий ровно по 1 разу через каждое ребро. Цикл называется простым, если он не проходит дважды через одно и то же ребро.





Эйлеровость

Лемма. Если степени всех вершин графа чётны и больше 0, то на графе существует простой цикл.

Доказательство

Выберем произвольную вершину в качестве начальной. Будем обходить граф, переходя из текущей вершины в соседнюю и удаляя ребро, по которому только что прошли. Т.к. степени всех вершин чётны, этот процесс остановится (мы не сможем идти дальше, т.к. у текущей вершины нет смежных) только тогда, когда мы вернёмся в начальную вершину. Таким образом, мы нашли цикл. В любом цикле мы можем выделить простой цикл, оставив только фрагмент между соседними вхождениями одной и той же вершины.

Лемма доказана.

Эйлеровость

Теорема. Связный граф является эйлеровым тогда и только тогда, когда степени всех его вершин чётны.

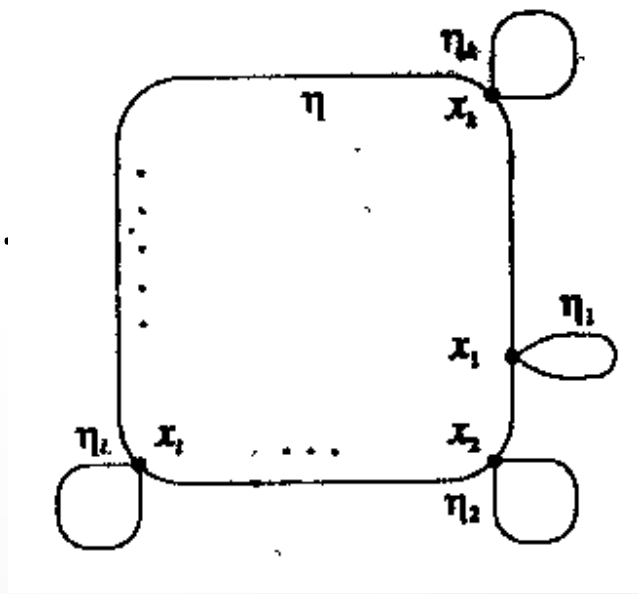
Доказательство

1) Если граф эйлеров, то на нём существует эйлеров цикл, и каждому прохождению цикла через каждую вершину соответствует ровно два ребра. Поэтому степени всех вершин чётны.
2) Если степени всех вершин чётны, то по лемме на графе есть простой цикл. Удалим этот цикл из графа. Для оставшегося графа степени всех вершин также будут чётными, поэтому каждая его компонента связности будет эйлеровым графом. Повторяя эту процедуру до тех пор, пока не удалим все рёбра, мы получим разбиение множества рёбер на простые циклы.

Эйлеровость

Эти простые циклы пересекаются друг с другом, т.к. иначе граф не связен. Выберем один из циклов и будем обходить его. Каждый раз, когда мы дошли до вершины *v*, в которой текущий цикл пересекается в другим циклом, мы начинаем обходить этот новый цикл. Обойдя его и вернувшись *v*, мы продолжаем обходить исходный цикл.

С помощью такой процедуры мы построим единый цикл, проходящий по всем вершинам.



Теорема доказана.

Эйлеровость

Определение. Связный граф называется *полуэйлеровым (квазиэйлеровым)*, если на нём существует простая *цепь*, проходящая ровно по 1 разу через каждое ребро.

Теорема. Связный граф является полуэйлеровым тогда и только тогда, когда степени всех его вершин, за исключением, возможно, ровно двух, чётны.

Сложность проверки графа на эйлеровость/полуэйлеровость (при использовании списков смежности): $O(n + m)$.

Нахождение эйлерова цикла

Имеется несколько алгоритмов построения эйлерова цикла. Рассмотрим наиболее известные. Во всех случаях предполагаем, что критерий эйлеровости заведомо выполнен.

Алгоритм Флёри (Fleury)

1. Выбрать произвольную вершину в качестве начала цикла.
2. Формировать цикл, переходя из текущей вершины в новую смежную с ней вершину, стирая пройденные рёбра. При этом если ребро – мост, то по нему идём только в случае отсутствия других возможностей.
3. Останавливаемся, когда больше нет возможности двигаться дальше.

Нахождение эйлерова цикла

Теорема. Если граф является эйлеровым, то алгоритм Флёри корректен, т.е. строит для данного графа эйлеров цикл.

Доказательство

1) Покажем, что процедура может выполняться на каждом шаге.

Пусть *s* – начальная вершина, *v* – текущая вершина. Тогда на текущем графе ровно 2 вершины нечётной степени (*s* и *v*), а степени остальных вершин чётны. Кроме того, текущий граф связен. Следовательно, текущий граф является полуэйлеровым, и на нём существует эйлеров путь из *s* в *v*. Поэтому описанная в алгоритме процедура всегда корректна.

Нахождение эйлерова цикла

2) Покажем, что результатом всегда является эйлеров цикл.

Это следует из того, что в силу чётности степеней всех вершин исходного графа, и его связности, алгоритм может завершиться только в начальной вершине.

Теорема доказана

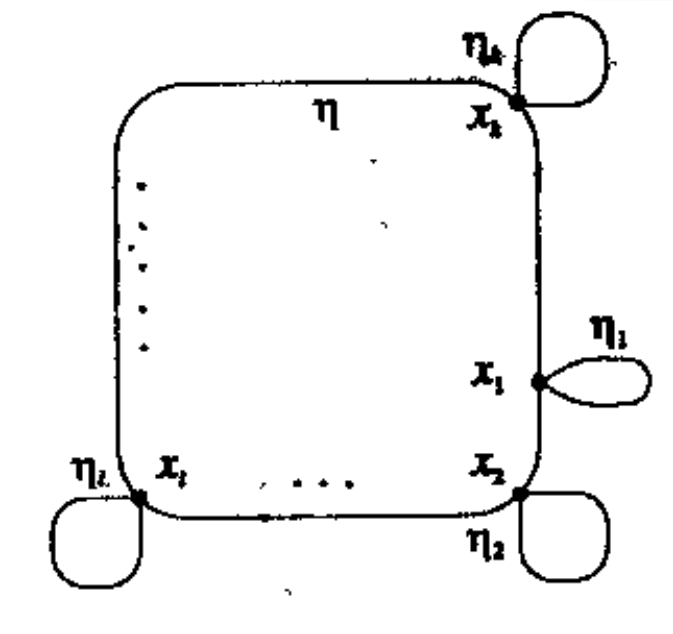
Временная сложность алгоритма Флёри: $O(m^2)$, поскольку для каждого из m рёбер необходимо проверять, является ли ребро мостом ($O(n+m)$).

Нахождение эйлерова цикла

Алгоритм Хирхольцера (Hierholzer)

Основан на применении стека и напоминает по структуре алгоритм поиска в глубину.

Идея алгоритма основана на процедуре, описанной в доказательстве критерия эйлеровости.



Нахождение эйлерова цикла

Алгоритм Хирхольцера (Hierholzer)

1. Создадим пустой стек и пустой список для хранения последовательности вершин цикла.
2. Выбрать произвольную вершину в качестве начала цикла.
3. Если для текущей вершины *u* есть инцидентное ребро (*u*,*v*), то
   1) Добавить текущую вершину *u* в стек.
   2) Вершину *v* сделать текущей.
   3) Удалить ребро (*u*,*v*).
4. Иначе (т.е. у текущей вершины *u* нет инцидентных рёбер):
   1) Добавить текущую вершину *u* в цикл.
   2) Если стек не пуст – извлечь вершину из стека и сделать её текущей. Иначе (т.е. если стек пуст) – останавливаемся.

Нахождение эйлерова цикла

Временная сложность алгоритма Хирхольцера: $O(m)$, поскольку алгоритм обрабатывает каждое ребро графа ровно один раз.