# Точки сочленения, мосты, компоненты связности

Виктор Васильевич Лепин

- Пусть $G = (V, E)$ связный неориентированный граф.

- Пусть $G = (V, E)$ связный неориентированный граф.
- Тоской сочленения в графе $G$ называется вершина, после удаления которой результирующий граф становится несвязным.

# Неориентированные графы

- Пусть $G = (V, E)$ связный неориентированный граф.
- Тоской сочленения в графе $G$ называется вершина, после удаления которой результирующий граф становится несвязным.
- Мостом в графе $G$ называется ребро, после удаления которого результирующий граф становится несвязным.

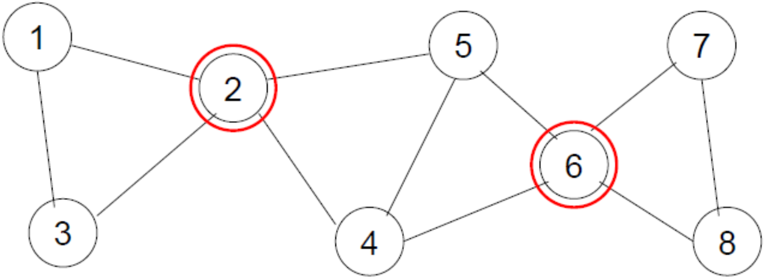
# Неориентированные графы

- Пусть $G = (V, E)$ связный неориентированный граф.
- Тоской сочленения в графе $G$ называется вершина, после удаления которой результирующий граф становится несвязным.
- Мостом в графе $G$ называется ребро, после удаления которого результирующий граф становится несвязным.
- Двусвязной компонентой графа $G$ называется максимальный по включению подграф $G'$ такой, что любые два ребры из $E(G')$ лежат на общем простом цикле.

# Неориентированные графы

- Пусть $G = (V, E)$ связный неориентированный граф.
- Тоской сочленения в графе $G$ называется вершина, после удаления которой результирующий граф становится несвязным.
- Мостом в графе $G$ называется ребро, после удаления которого результирующий граф становится несвязным.
- Двусвязной компонентой графа $G$ называется максимальный по включению подграф $G'$ такой, что любые два ребры из $E(G')$ лежат на общем простом цикле.
- Если граф моделирует физическую сеть, для которой возможны разрушения ее элементов, то эти понятия весьма важны.

# Точки сочленения



# Как найти все точки сочленения?

- Подход грубой силы: испытывайте одну за другой вершины; удалите испытываемую вершину и смотрите, не приводит ли удаление вершины к несвязному графу:

# Как найти все точки сочленения?

- Подход грубой силы: испытывайте одну за другой вершины; удалите испытываемую вершину и смотрите, не приводит ли удаление вершины к несвязному графу:
    - Для каждой вершины $v$ выполните:

## Как найти все точки сочленения?

- Подход грубой силы: испытывайте одну за другой вершины; удалите испытываемую вершину и смотрите, не приводит ли удаление вершины к несвязному графу:
  - Для каждой вершины $v$ выполните:
  - удалите $v$ из графа: $G' := G - v$;

# Как найти все точки сочленения?

- Подход грубой силы: испытывайте одну за другой вершины; удалите испытываемую вершину и смотрите, не приводит ли удаление вершины к несвязному графу:
  - Для каждой вершины $v$ выполните:
  - удалите $v$ из графа: $G' := G - v$;
  - Используя поиск в глубину, или поиск в ширину, проверте является ли граф $G'$ связным?

# Как найти все точки сочленения?

- Подход грубой силы: испытывайте одну за другой вершины; удалите испытываемую вершину и смотрите, не приводит ли удаление вершины к несвязному графу:
  - Для каждой вершины $v$ выполните:
  - удалите $v$ из графа: $G' := G - v$;
  - Используя поиск в глубину, или поиск в ширину, проверте является ли граф $G'$ связным?
  - Если он несвязен, то добавте $v$ в множество $AP$ точек сочленения.

# Как найти все точки сочленения?

- Подход грубой силы: испытывайте одну за другой вершины; удалите испытываемую вершину и смотрите, не приводит ли удаление вершины к несвязному графу:
  - Для каждой вершины $v$ выполните:
  - удалите $v$ из графа: $G' := G - v$;
  - Используя поиск в глубину, или поиск в ширину, проверте является ли граф $G'$ связным?
  - Если он несвязен, то добавте $v$ в множество $AP$ точек сочленения.
- Временная сложность вышеуказанного метода составляет $O(n(n+m))$ для графа, представленного с использованием списков смежности.

# Как найти все точки сочленения?

- Подход грубой силы: испытывайте одну за другой вершины; удалите испытываемую вершину и смотрите, не приводит ли удаление вершины к несвязному графу:
  - Для каждой вершины $v$ выполните:
  - удалите $v$ из графа: $G' := G - v$;
  - Используя поиск в глубину, или поиск в ширину, проверте является ли граф $G'$ связным?
  - Если он несвязен, то добавте $v$ в множество $AP$ точек сочленения.
- Временная сложность вышеуказанного метода составляет $O(n(n+m))$ для графа, представленного с использованием списков смежности.
- Можно ли найти множество $AP$ быстрее?

## Как найти все точки сочленения? Использовать поиск в глубину.

Можно доказать следующие свойства дерева поиска в глубину:

# Как найти все точки сочленения? Использовать поиск в глубину.

Можно доказать следующие свойства дерева поиска в глубину:

- Корень DFS-дерева является точкой сочленения тогда и только тогда, когда у него есть не менее двух сыновей.

# Как найти все точки сочленения? Использовать поиск в глубину.

Можно доказать следующие свойства дерева поиска в глубину:

- Корень DFS-дерева является точкой сочленения тогда и только тогда, когда у него есть не менее двух сыновей.
- Некорневая вершина $v$ DFS-дерева является точкой сочленения в $G$ тогда и только тогда, когда она имеет сына $s$, такого, что нет обратного ребра от $s$ или от любого потомка $s$ к предку $v$.

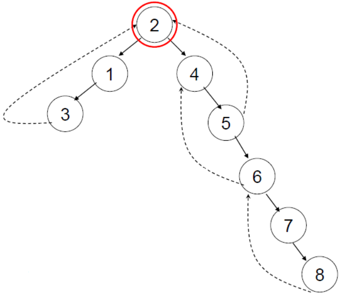
# Как найти все точки сочленения? Использовать поиск в глубину.

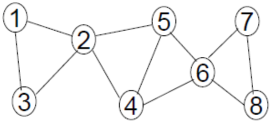
Можно доказать следующие свойства дерева поиска в глубину:

- Корень DFS-дерева является точкой сочленения тогда и только тогда, когда у него есть не менее двух сыновей.
- Некорневая вершина $v$ DFS-дерева является точкой сочленения в $G$ тогда и только тогда, когда она имеет сына $s$, такого, что нет обратного ребра от $s$ или от любого потомка $s$ к предку $v$.
- Листья DFS-дерева никогда не являются точками сочленения.

# Использование поиска в глубину.

Корень DFS-дерева является точкой сочленения тогда и только тогда, когда у него есть не менее двух сыновей.

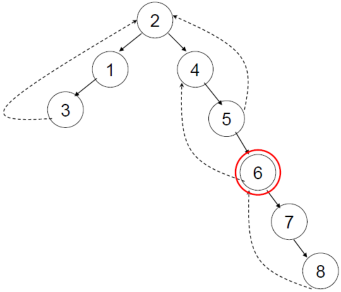


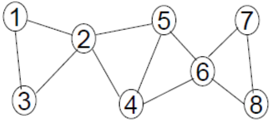




Узел 2 является ТС, потому что любой узел из первого поддерева (1, 2) соединен с любым узлом из второго поддерева (4, 5, 6, 7, 8) путем, который включает узел 2. Если узел 2 удалить, то поддеревья становятся несвязными.

# Использование поиска в глубину.

Некорневой узел $v$ DFS-дерева является точкой сочленения тогда и только тогда, когда он не имеет сына связанного (непосредственно или через потомка) с предком $v$.




Узел 6 является ТС, потому что его дочерний узел 7 не связан обратными ребрами с предком 6.

## Использование поиска в глубину.

- Вершины изначально окрашены в белый цвет (стек пуст).

# Использование поиска в глубину.

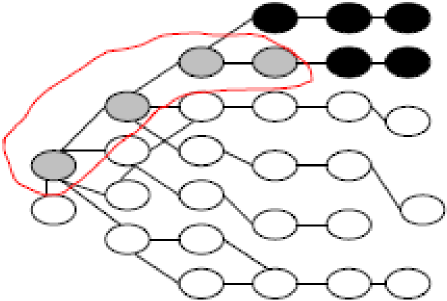
- Вершины изначально окрашены в белый цвет (стек пуст).
- Затем вершина окрашивается в серый цвет при первом поподании в неё (она заносится в стек).

# Использование поиска в глубину.

- Вершины изначально окрашены в белый цвет (стек пуст).
- Затем вершина окрашивается в серый цвет при первом поподании в неё (она заносится в стек).
- Затем — в черный, когда ее исследование закончено (она удаляется из стека).

# Использование поиска в глубину.

- Вершины изначально окрашены в белый цвет (стек пуст).
- Затем вершина окрашивается в серый цвет при первом поподании в неё (она заносится в стек).
- Затем — в черный, когда ее исследование закончено (она удаляется из стека).



# Использование поиска в глубину.

Пусть каждая вершина $v$ имеет следующие атрибуты:

# Использование поиска в глубину.

Пусть каждая вершина $v$ имеет следующие атрибуты:

- v.color: (white, grey, black) — ее цвет;
- v.pi представляет «родительский» узел $v$

# Использование поиска в глубину.

Пусть каждая вершина $v$ имеет следующие атрибуты:

- v.color: (white, grey, black) — ее цвет;
- v.pi представляет «родительский» узел $v$
- v.d представляет момент времени, когда впервые перешли в вершину $v$ (она заталкивается в стек)

# Использование поиска в глубину.

Пусть каждая вершина $v$ имеет следующие атрибуты:

- v.color: (white, grey, black) — ее цвет;
- v.pi представляет «родительский» узел $v$
- v.d представляет момент времени, когда впервые перешли в вершину $v$ (она заталкивается в стек)
- v.f представляет момент времени, когда исследование вершины $v$ закончено (она удалена из стека)

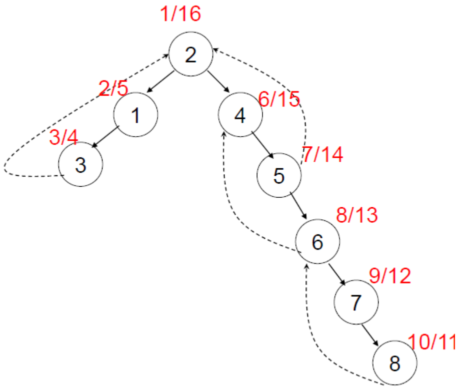
# Использование поиска в глубину.

**DFS(G)**

```
for each vertex u ∈ G.V
    u.color:=WHITE;
    u.p:=NIL;
time:=0;
for each vertex u ∈ G.V
    if u.color=WHILE
        DFS-Visit(G,u);
```

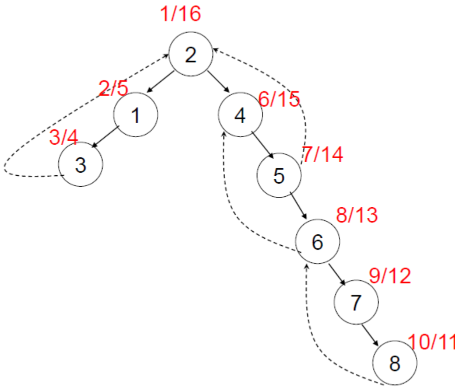
**DFS-Visit(G,u)**

```
time:=time+1;
u.d:=time;
u.color:=GRAY;
for each v ∈ G.Adj[u];
    if v.color=WHITE
        v.p:=u;
        DFS-Visit(G,v)
u.color:=BLACK;
time:=time+1
u.f:=time
```



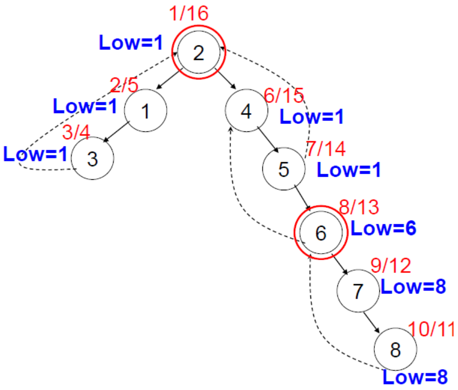

Момент времени входа (v.d) в узел всегда меньше момента времени входа в любого потомка этой вершины в DFS-дереве.

# Использование поиска в глубину.




Момент времени входа (v.d) в узел всегда меньше момента времени входа в любого потомка этой вершины в DFS-дереве.
Обратные ребра ведут к узлам, имеющим меньший момент времени входа.

LOW (u) = моменту времени входа в вершину v, являющуюся наивысшим предком u, которого можно достичь от u, или потомка u с помощью обратных ребер.




Вершина $u$ является точкой сочленения тогда и только тогда, когда она имеет потомка $v$ с $\text{LOW}(v) \geq u.d$

Элементы алгоритма

- Во время DFS вычислите также значения функции LOW для каждой вершины.

Элементы алгоритма

- Во время DFS вычислите также значения функции LOW для каждой вершины.
- После завершения рекурсивного поиска от сына v вершины $u$ мы обновляем $u.low$ значением $v.low$. Вершина $u$ является точкой сочленения, если $v.low \geq u.d$.

Элементы алгоритма

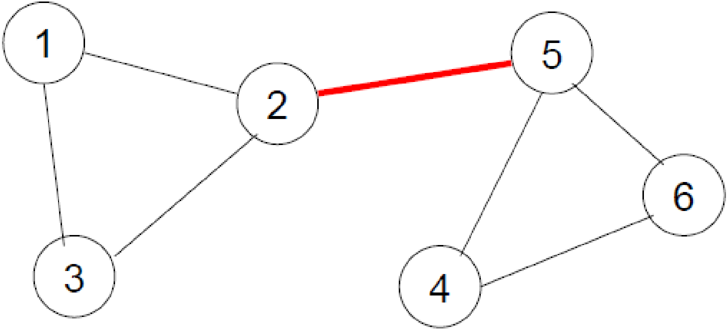
- Во время DFS вычислите также значения функции LOW для каждой вершины.
- После завершения рекурсивного поиска от сына v вершины $u$ мы обновляем $u.low$ значением $v.low$. Вершина $u$ является точкой сочленения, если $v.low \geq u.d$.
- Если вершина $u$ является корнем DFS-дерева, проверьте, является ли $v$ его не первым сыном.

Элементы алгоритма

- Во время DFS вычислите также значения функции LOW для каждой вершины.
- После завершения рекурсивного поиска от сына v вершины $u$ мы обновляем $u.low$ значением $v.low$. Вершина $u$ является точкой сочленения, если $v.low \geq u.d$.
- Если вершина $u$ является корнем DFS-дерева, проверьте, является ли $v$ его не первым сыном.
- Если обнаружено обратное ребро $(u, v)$, то обновите $u.low$ значением $v.d$.

```
DFS_VISIT_AP(G, u)
      time=time+1
      u.d=time
      u.color=GRAY
      u.low=u.d
      for each v in G.Adj[u]
            if v.color==WHITE
                  v.pi=u
                  DFS_VISIT_AP(G,v)
                  if (u.pi==NIL)
                        if (v is second son of u)
                              "u is AP"   // Case 1
                  else
                        u.low=min(u.low, v.low)
                        if (v.low>=u.d)
                               "u is AP"   // Case 2
            else  if ((v<>u.pi) and (v.d <u.d))
                        u.low=min(u.low, v.d)
      u.color=BLACK
      time=time+1
      u.f=time
```

# Пример моста



# Как найти множество мостов

- Подход грубой силы: испытывайте одно за другим ребро; удалите испытываемое ребро и смотрите, не приводит ли удаление ребра к несвязному графу:

# Как найти множество мостов

- Подход грубой силы: испытывайте одно за другим ребро; удалите испытываемое ребро и смотрите, не приводит ли удаление ребра к несвязному графу:
  - Для каждого ребра $vu$ выполните:

# Как найти множество мостов

- Подход грубой силы: испытывайте одно за другим ребро; удалите испытываемое ребро и смотрите, не приводит ли удаление ребра к несвязному графу:
  - Для каждого ребра $vu$ выполните:
  - удалите $vu$ из графа: $G' := G - vu$;

# Как найти множество мостов

- Подход грубой силы: испытывайте одно за другим ребро; удалите испытываемое ребро и смотрите, не приводит ли удаление ребра к несвязному графу:
    - Для каждого ребра $vu$ выполните:
    - удалите $vu$ из графа: $G' := G - vu$;
    - Используя поиск в глубину, или поиск в ширину, проверте является ли граф $G'$ связным?

# Как найти множество мостов

- Подход грубой силы: испытывайте одно за другим ребро; удалите испытываемое ребро и смотрите, не приводит ли удаление ребра к несвязному графу:
    - Для каждого ребра $vu$ выполните:
    - удалите $vu$ из графа: $G' := G - vu$;
    - Используя поиск в глубину, или поиск в ширину, проверте является ли граф $G'$ связным?
    - Если он несвязен, то добавте $vu$ в множество $B$ мостов.

# Как найти множество мостов

- Подход грубой силы: испытывайте одно за другим ребро; удалите испытываемое ребро и смотрите, не приводит ли удаление ребра к несвязному графу:
  - Для каждого ребра $vu$ выполните:
  - удалите $vu$ из графа: $G' := G - vu$;
  - Используя поиск в глубину, или поиск в ширину, проверте является ли граф $G'$ связным?
  - Если он несвязен, то добавте $vu$ в множество $B$ мостов.
- Временная сложность вышеуказанного метода составляет $O(m(n + m))$ для графа, представленного с использованием списков смежности.

# Как найти множество мостов

- Подход грубой силы: испытывайте одно за другим ребро; удалите испытываемое ребро и смотрите, не приводит ли удаление ребра к несвязному графу:
  - Для каждого ребра $vu$ выполните:
  - удалите $vu$ из графа: $G' := G - vu$;
  - Используя поиск в глубину, или поиск в ширину, проверте является ли граф $G'$ связным?
  - Если он несвязен, то добавте $vu$ в множество $B$ мостов.
- Временная сложность вышеуказанного метода составляет $O(m(n+m))$ для графа, представленного с использованием списков смежности.
- Можно ли найти множество $B$ быстрее?

# Поиск мостов поиском в глубину

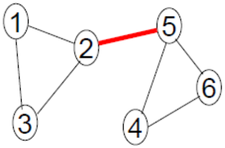
- Ребро графа $G$ является мостом тогда и только тогда, когда оно не лежит ни на одном простом цикле $G$.

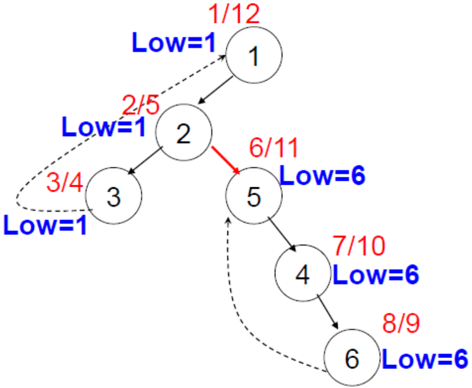
# Поиск мостов поиском в глубину

- Ребро графа $G$ является мостом тогда и только тогда, когда оно не лежит ни на одном простом цикле $G$.
- Если в некоторую вершину $u$ входит обратное ребро, то никакое ребро ниже $u$ в DFS-дереве не может быть мостом. Причина в том, что каждое обратное ребро дает нам цикл, и никакое ребро, которое является членом цикла, не может быть мостом.

- Ребро графа $G$ является мостом тогда и только тогда, когда оно не лежит ни на одном простом цикле $G$.
- Если в некоторую вершину $u$ входит обратное ребро, то никакое ребро ниже $u$ в DFS-дереве не может быть мостом. Причина в том, что каждое обратное ребро дает нам цикл, и никакое ребро, которое является членом цикла, не может быть мостом.
- Если у нас есть вершина $v$, и $u$ — ее родитель в DFS-дереве и ни один из предков $v$ не имеет обратного ребра, входящего в нее, то $(u, v)$ является мостом.

# Поиск мостов поиском в глубину



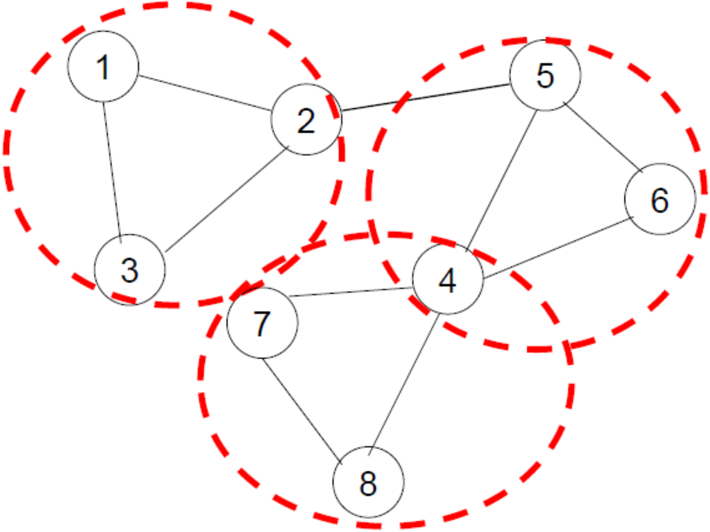

Ребро $(u, v)$ является мостом, тогда и только тогда, когда $\mathrm{LOW}(v) > u.d$.

# Поиск мостов поиском в глубину

```
DFS_VISIT_Bridges(G, u)
      time=time+1
      u.d=time
      u.color=GRAY
      u.low=u.d
      for each v in G.Adj[u]
            if v.color==WHITE
                  v.pi=u
                  DFS_VISIT_AP(G,v)

                  u.low=min(u.low, v.low)
                  if (v.low>u.d)
                               "(u,v) is Bridge"
            else  if ((v<>u.pi) and  (v.d <u.d))
                        u.low=min(u.low, v.d)
      u.color=BLACK
      time=time+1
      u.f=time
```

# Пример двусвязных компонент



# Поиск двусвязных компонент

- Две двусвязные компоненты не могут иметь общего ребра, но могут иметь общую вершину.

# Поиск двусвязных компонент

- Две двусвязные компоненты не могут иметь общего ребра, но могут иметь общую вершину.
  - Припишим концевым вершинам ребра идентификатор их двусвязной компоненты.

# Поиск двусвязных компонент

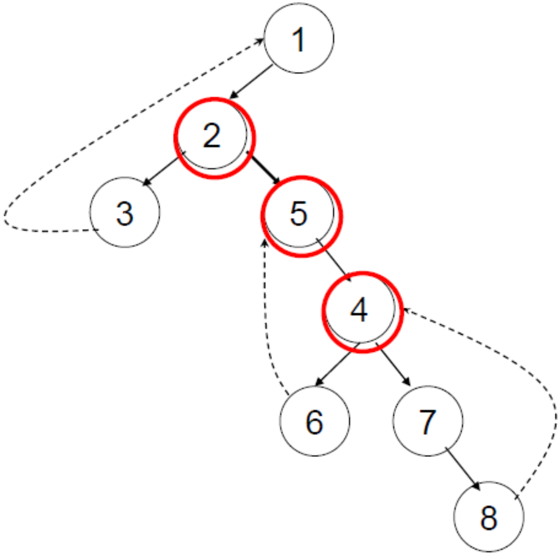
- Две двусвязные компоненты не могут иметь общего ребра, но могут иметь общую вершину.
  - Припишим концевым вершинам ребра идентификатор их двусвязной компоненты.
- Общая вершина нескольких двусвязных компонент является точкой сочленения.

# Поиск двусвязных компонент

- Две двусвязные компоненты не могут иметь общего ребра, но могут иметь общую вершину.
  - Припишим концевым вершинам ребра идентификатор их двусвязной компоненты.
- Общая вершина нескольких двусвязных компонент является точкой сочленения.
- Точки сочленения разделяют двусвязные компоненты графа. Если в графе нет точек сочленения, то он является двусвязным.

# Поиск двусвязных компонент

- Две двусвязные компоненты не могут иметь общего ребра, но могут иметь общую вершину.
    - Припишим концевым вершинам ребра идентификатор их двусвязной компоненты.
- Общая вершина нескольких двусвязных компонент является точкой сочленения.
- Точки сочленения разделяют двусвязные компоненты графа. Если в графе нет точек сочленения, то он является двусвязным.
    - Попытаемся идентифицировать двусвязные компоненты при поиске точек сочленения.


1
2
3
5
4
6
7
8

Элементы алгоритма

- Во время DFS используйте стек для хранения посещенных ребер (ребер дерева или обратных ребер).

# Поиск двусвязных компонент

Элементы алгоритма

- Во время DFS используйте стек для хранения посещенных ребер (ребер дерева или обратных ребер).
- После того, как мы закончим рекурсивный поиск для сына $v$ вершины $u$, мы проверяем, является ли $u$ точкой сочленения для $v$. Если это так, мы выводим все ребра из стека до $(u, v)$. Эти ребра порождают двусвязную компоненту.

# Поиск двусвязных компонент

Элементы алгоритма

- Во время DFS используйте стек для хранения посещенных ребер (ребер дерева или обратных ребер).
- После того, как мы закончим рекурсивный поиск для сына $v$ вершины $u$, мы проверяем, является ли $u$ точкой сочленения для $v$. Если это так, мы выводим все ребра из стека до $(u, v)$. Эти ребра порождают двусвязную компоненту.
- Когда мы вернемся к корню DFS-дерева, мы должны вывести рёбра, даже если корень не является точкой сочленения (граф может быть двусвязным) — мы не будем проверять случай, когда корень является точкой сочленения.

# Поиск двусвязных компонент

```
DFS_VISIT_BiconnectedComp(G, u)
        time=time+1
        u.d=time
        u.color=GRAY
        u.low=u.d
        u.AP=false
        for each v in G.Adj[u]
                if v.color==WHITE
                        v.pi=u
                        EdgeStack.push(u,v)
                        DFS_VISIT_AP(G,v)
                        u.low=min(u.low, v.low)
                        if (v.low>=u.d)
                                pop all edges from EdgeStack until (u,v)
                                these are the edges of a Biconn Comp
                else    if ((v<>u.pi) and (v.d <u.d))
                                EdgeStack.push(u,v)
                                u.low=min(u.low, v.d)

        u.color=BLACK
        time=time+1
        u.f=time
```