Введение в дискретную математику и математическую логику

Лекция №7

Двусвязность

- Апанович Зинаида Владимировна

ДВУСВЯЗНОСТЬ

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

Вершина *a* называется точкой сочленения графа *G*, если существуют вершины *v* и *w* такие, что *v, w* и *a* различны, и каждый путь между *v* и *w* содержит вершину *a* .

Другими словами, *a* является точкой сочленения *G*, если удаление *a* разделяет *G* на ≥ 2 части.

Определение. Граф *G* двусвязен, если для каждой тройки вершин *v, w, a* существует путь между *v* и *w,* не содержащий *a* .

Таким образом, неориентированный связный граф является двусвязным $\Leftrightarrow$ он не имеет точек сочленения.

Двусвязная компонента (блок)

Мы можем определить естественное отношение на множестве ребер графа G, сказав, что 2 ребра e_1 и e_2 находятся в отношении R, если

$e_1 = e_2$ или существует цикл, содержащий e_1 и e_2 .

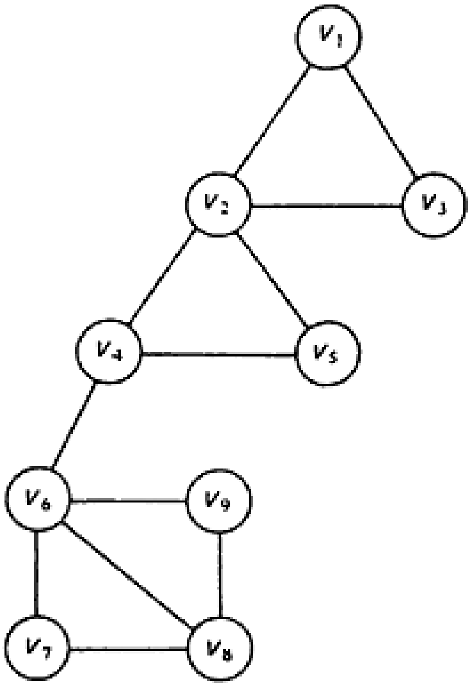
Легко показать, что это отношение является отношением эквивалентности, которое разбивает ребра графа G на классы эквивалентности $E_1, E_2, \ldots, E_k$ таким образом, что 2 различных ребра находятся в одном классе $\Leftrightarrow$ они лежат на общем цикле.

Для $1 \leq i \leq k$, пусть V_i будет множеством инцидентных вершин для ребер во множестве E_i .

Каждый граф $G_i = (V_i, E_i)$ называется двусвязной компонентой (блоком) графа G .

Двусвязная компонента (блок)

Пример. Рассмотрим неориентированный граф ниже. Сколько двусвязных компонент показано?



Двусвязная компонента (блок)

Решение Имеется 4 двусвязных компоненты.

Вершина v_4, например, является точкой сочленения, поскольку каждый путь между v_1 и v_7 проходит через v_4

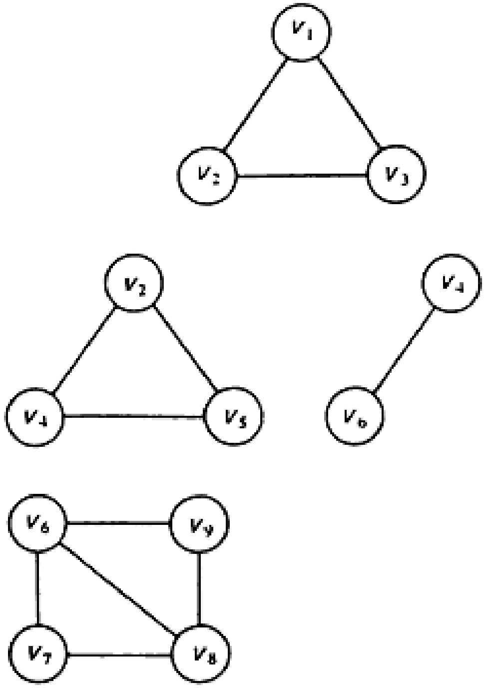
Классы эквивалентности ребер, лежащих на общих циклах, равны

$\{\{ v_1 , v_2 \}, \{ v_1 , v_3 \}, \{ v_2 , v_3 \}\}$,

$\{\{ v_2 , v_4 \}, \{ v_2 , v_5 \}, \{ v_4 , v_5 \}\}$,

$\{\{v_4 , v_6 \}\}$,

$\{\{v_6 , v_7 \}, \{ v_6 , v_8 \}, \{ v_6 , v_9 \}, \{ v_7 , v_8 \}, \{ v_8 , v_9 \}\}$.



Свойства двусвязных компонент

Лемма 1. Для $1 \leq i \leq k$, пусть $G_i = (V_i\ , E_i)$ будут двусвязными компонентами связного неориентированного графа $G = (V, E)$.

Тогда

1. G_i двусвязен для каждого i , $1 \leq i \leq k$.
2. Для всех $i \neq j$, $V_i \cap V_j$ содержит не более 1 вершины.
3. a — точка сочленения $G \Leftrightarrow a \in G_i \cap G_j$ для некоторых $i \neq j$.

Свойства двусвязных компонент

1) G_i двусвязен для каждого i, $1 \le i \le k$.

Доказательство (от противного)

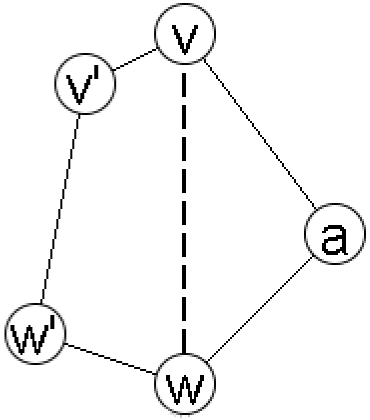
Предположим, что в V есть 3 различные вершины v, w и a , такие, что все пути в G_i между v и w проходят через a .

Тогда, конечно, (v, w) не является ребром в E_i . Таким образом, существуют различные ребра (v, v') и (w, w') в E_i , и существует цикл в G_i , содержащий эти ребра.

По определению двусвязной компоненты все ребра и вершины этого цикла находятся в E_i и V_i соответственно.

Значит, в G_i есть 2 пути между v и w, только 1 из которых может содержать a,

противоречие .



Свойства двусвязных компонент

2. Для всех $i \neq j$, $V_i \cap V_j$ содержит не более 1 вершины.

Предположим, что имеются 2 различные вершины v и w, принадлежащие $V_i \cap V_j$.

Тогда существует цикл C_1 в G_i , содержащий v и w, и цикл C_2 в G_j , также содержащий v и w .

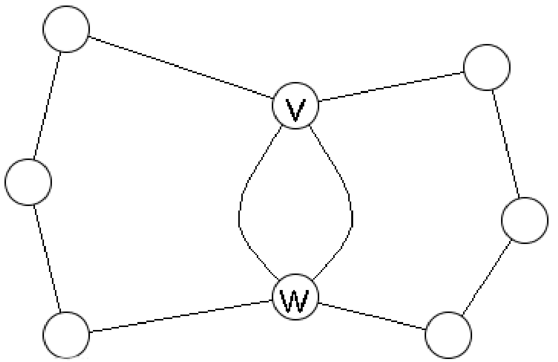
Так как E_i и E_j не пересекаются, то множества ребер в C_1 и C_2 не пересекаются.

Однако мы можем построить цикл, содержащий v и w , который использует ребра как из C_1, так и *из* C_2, и, значит, по крайней мере 1 ребро в E_i эквивалентно ребру в E_j .

Таким образом, E_i и E_j не являются

классами эквивалентности,

как предполагалось.



Свойства двусвязных компонент

3. a — точка сочленения G $\Leftrightarrow$ $a \in V_i \cap V_j$ для некоторых $i \neq j$.

=> Предположим , что вершина a является точкой сочленения G.

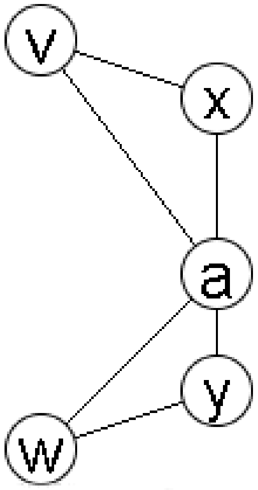
Тогда существуют 2 вершины v и w такие, что v, w и a различны, и каждый путь между v и w содержит a .

Поскольку G связен, существует по крайней мере 1 такой путь.

Пусть $\{ x , a \}$ и $\{ y , a \}$ — два ребра на пути между v и w, инцидентные a .

Если существует цикл, содержащий эти 2 ребра, то существует путь между v и w , не содержащий a .

Таким образом, $\{ x , a \}$ и $\{ y , a \}$ находятся в различных двусвязных компонентах , а a находится в пересечении множеств их вершин.

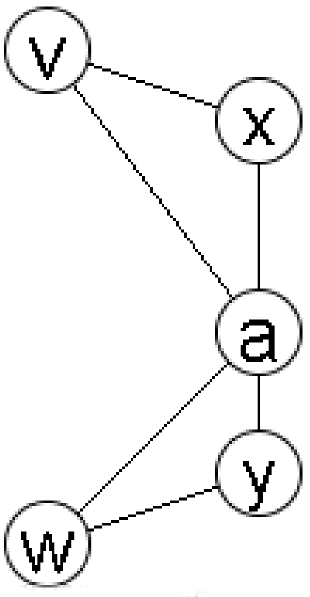


Свойства двусвязных компонент

< = Если $a \in V_i \cap V_j$, тогда есть ребра { x, a } и { y, a } в E_i и E_j соответственно.

Поскольку оба эти ребра не принадлежат одному циклу, то они лежат в разных циклах цикле, и каждый путь их x в y содержит a .

Таким образом, a является точкой сочленения .



Поиск в глубину и точки сочленения

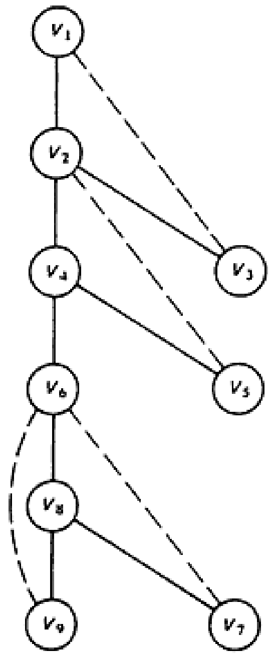
Поиск в глубину особенно полезен при нахождении точек сочленения и двусвязных компонент неориентированного графа .

Одной из причин этого является отсутствие «поперечных ребер».

То есть, если вершина v не является ни предком, ни потомком вершины w в остовном лесу, то не может быть ребра из v в w .

Поиск в глубину и точки сочленения

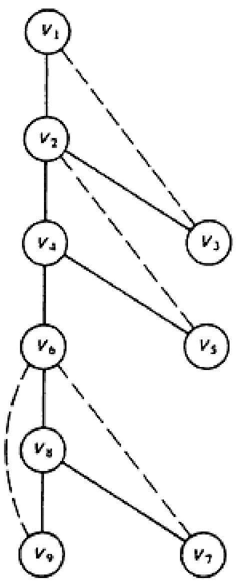
- Если вершина *a* является точкой сочленения, то удаление *a* и всех ребер, инцидентных *a*, *разбивает* граф на 2 или более частей.
- Одна часть состоит из сына *a* и *всех* его потомков в дереве поиска в глубину.
- Таким образом, в дереве поиска в глубину *a* должна иметь сына *s* , такого, что не существует обратного ребра между потомком *s* и собственным предком *a* .



Поиск в глубину и точки сочленения

И обратно, за исключением корня остовного дерева, отсутствие поперечных ребер подразумевает, что вершина *a* является точкой сочленения, если нет обратного ребра от любого потомка некоторого сына *a* к *правильному предку a*.

Корень остовного дерева поиска в глубину является точкой сочленения $\Leftrightarrow$ у него есть ≥ 2 сына.



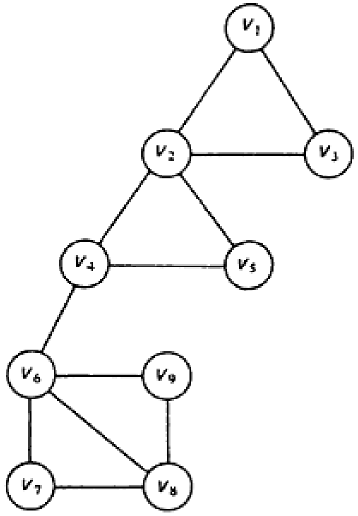
Поиск в глубину и точки сочленения

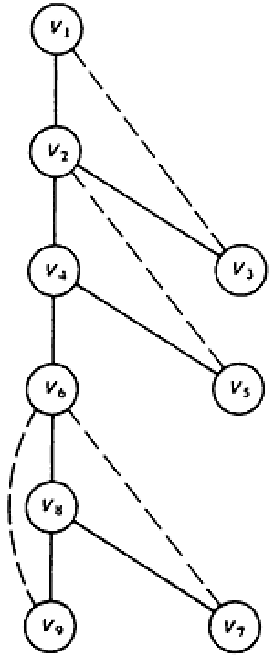
Пример. Глубинное остовное дерево для графа показано ниже.

Точки сочленения — v_2, v_4 и v_6.

У вершины v_2 есть сын v_4, и ни один потомок v_4 не имеет обратных ребер к собственному предку v_2.

Аналогично, у вершины v_4 есть сын v_6, и у вершины v_6 есть сын v_8 с аналогичным свойством.





Поиск в глубину и точки сочленения

Перечисленные идеи воплощены в следующей лемме.

Лемма 2. Пусть $G = (V, E)$ — связный неориентированный граф, а $S = (V, T)$ — дерево поиска в глубину для G.

Вершина a является точкой сочленения G $\Leftrightarrow$ либо

(1) a является корнем и a имеет > 1 сына,

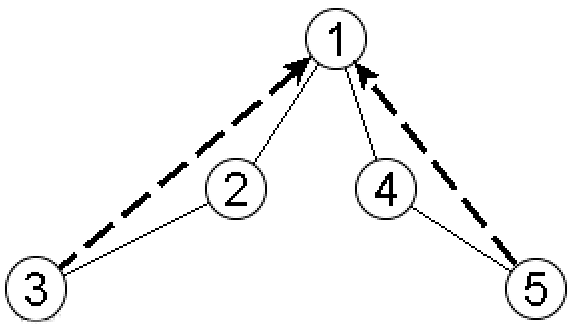
или

(2) *a не* является корнем и для некоторого сына s вершины a *нет* обратного ребра между любыми *потомками* из s (включая сам s) и правильным предком a.

Поиск в глубину и точки сочленения

Легко показать, что корень является точкой сочленения ⇔ у него > 1 сына.

(Упражнение).



Поиск в глубину и точки сочленения

$\Rightarrow$ Предположим, что *условие* 2 истинно

(вершина *a не* является корнем и для некоторого сына вершины *a* нет обратного ребра между каким- либо *потомком s* и *собственным* предком *a*).

Пусть *p* будет родителем *a*. Каждое обратное ребро идет от вершины к ее предку.

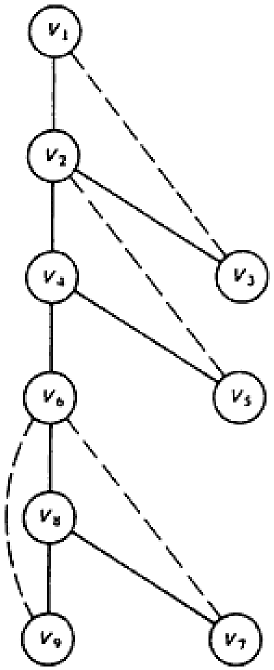
Таким образом, любое обратное ребро *от* потомка *s* вершины *v* должно идти к предку *s*.

По условию леммы обратное ребро не может идти к собственному предку *a.*

Следовательно, оно идет либо *к a* или

потомку *a*.

Таким образом, каждый путь от *s* до *p* содержит *a*, и , значит, *a* является точкой сочленения .



Поиск в глубину и точки сочленения

<= Чтобы доказать обратное, предположим, что a является точкой сочлененя, но не корнем .

Пусть x и y — различные вершины, отличные от a, такие, что каждый путь в G между x и y содержит a .

По *крайней мере одна из* x *и* y*, скажем* x *, является правильным потомком* a в S, в противном случае существует путь в G между x и y, использующий ребра в T и избегающий a .

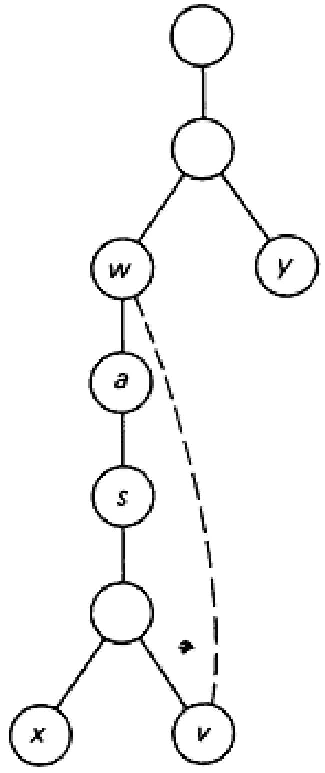
Пусть s *будет* таким сыном, что x является потомком s (возможно, $x = s$).

Возможны 2 варианта.

Либо нет обратного ребра между потомком v вершины s и собственным предком w вершины a, в этом случае Лемма 2 сразу же становится истинной,

или такое ребро { v, w } существует.

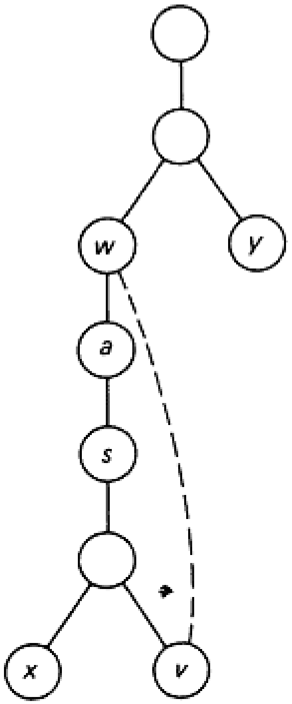
В последнем случае следует рассмотреть два случая.



(a)

СЛУЧАЙ 1. Предположим *, что у* не является потомком *а.*

Тогда существует путь от *x в v* и в *y* , который не проходит через *а*, противоречие.



(a)

Поиск в глубину и точки сочленения

СЛУЧАЙ 2. Предположим, что y является потомком a .

Конечно, y не является потомком s, иначе существует путь от x к y , который обходит a .

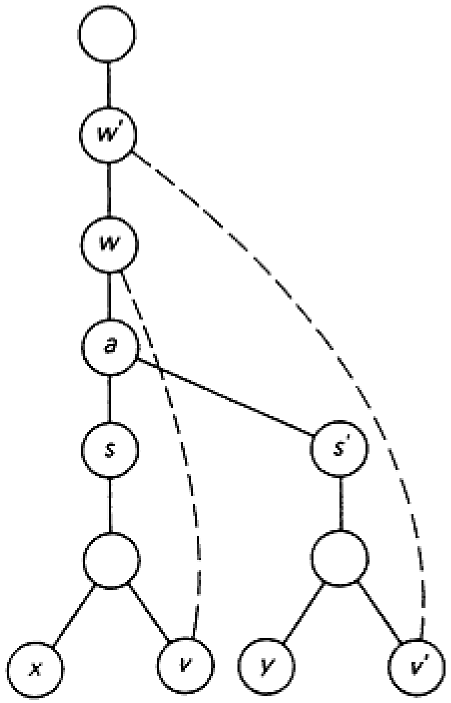
Пусть s' будет сыном вершины a, таким, что y является потомком s.

Опять возможны 2 варианта. Либо нет обратного ребра между потомком v вершины s' и собственным предком w вершины a , в этом случае лемма 2 сразу же становится истинной,

или такое ребро (v', w') существует.

В последнем случае имеется путь из x в v, w, w', v , y , что позволяет не проходить через a. *Противоречие* .

Приходим к выводу, что Лемма 2 верна.



Понятие *v.LOW*

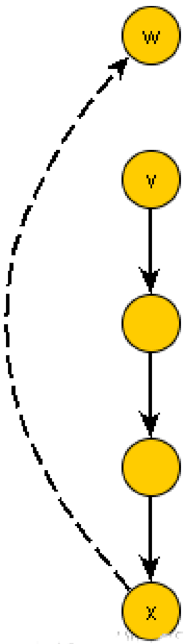
Пусть T и B — множества деревесных и обратных ребер, полученные путем поиска в глубину на связном неориентированном графе $G = (V, E)$.

Мы предполагаем, что вершины в V проименованы по их номерам поиска в глубину ($v.d$).

Для каждой v в V мы определяем

$v.LOW$ = MIN({v} U {w|существует обратное ребро $\{ x, w \} \in B$

- такое, что x является потомком v,
- а w предком v в глубинном остовном лесу (V , T)}) (1)



Нумерация $v.d$ подразумевает, что если x является потомком из v и $\{ x , w \}$ — обратное ребро такое, что $w < v$, то w является собственным предком v.

Таким образом, по лемме 2, если вершина v *не* является корнем T, то v является *точкой сочленения* $\Leftrightarrow$ у v есть сын s такой что

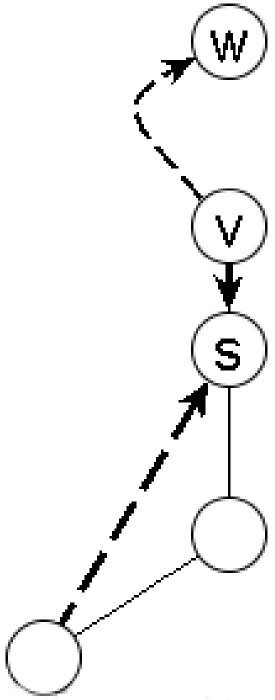
$s.Low \geq v.$

Понятие *v.LOW*

Мы можем внедрить в процедуру DFS_visit вычисление для определения значения *Low* каждой вершины, если перепишем выражение (1) так, чтобы выразить *v.Low* через вершины, смежные с *v* , через обратные ребра и значения *Low* в сыновьях *v* .

В частности, *v.Low* можно вычислить, определив минимальное значение таких вершин *w*, что либо

1. w = *v*, или
2. *w* = *s.Low* и *s* является сыном *v*, или
3. (*v*, *w*) —обратное ребро в *B* .

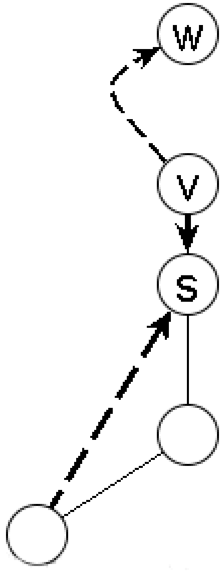


Понятие *v.LOW*

Минимальное значение *w* можно определить, как только список вершин, смежных с *v*, Adj[*v*], будет исчерпан.

Таким образом, (1) эквивалентно

$v.Low$ = MIN({ v } U { $s.Low$ | s является сыном v } U { w |{ v, w } $\in B$ }).   (2)



DFS_visit с вычислением LOW

Процедура *DFS _ Visit _Biconn(v);*

v.color = GRAY ;

v.d = time;

time = time +1;

v.Low = v.d ;

DFS_visit с вычислением LOW (продолжение)

```
for each vertex w ∈ Adj[v] do {
    if w.color == "WHITE" then{ /* (v, w) – древесное ребро*/
        w.π = v /* add (v, w) to T/
        DFS_Visit_Biconn(w);
        if w.Low ≥ v.d  then /* найдена двусвязная компонента*/
            v.Low = MIN(v.Low, w.Low);
       } /* конец обработки древесного ребра*/
   else if v.π != w then /*(v, w) – обратное ребро*/
            v.Low =MIN(v.Low, w.d);
 }
}}
}
```

DFS_visit с вычислением LOW

Мы включили как переименование вершин по первому посещению, так и вычисление *Low* в пересмотренную версию DFS_visit.

Сначала мы инициализируем *v.Low* до максимально возможного значения.

Если вершина *v* имеет сына *w* в глубинном остовном лесу, то мы корректируем *v.Low* , если *w.Low* < *v.Low* .

Если вершина *v* соединена обратным ребром с вершиной *w*, то мы делаем *v.Low равным w.d , если номер при поиске в* глубину вершины *w* меньше текущего значения *v.Low* .

Тест проверяет случай, когда (*v, w*) на самом деле не является обратным ребром, поскольку *w* является отцом *v в* глубинном остовном дереве.

Найдя *v*. *Low* для каждой вершины *v,* мы можем легко определить точки сочленения.

Алгоритм 3. Нахождение двусвязных компонент.

Вход. Связный неориентированный граф $G = (V, E)$.

Выход . Список ребер каждой двусвязной компоненты графа G.

1. Изначально установить $T = \varnothing$, а time = 0.

Также пометить каждую вершину в V как «БЕЛУЮ».

Затем выбрать произвольную вершину v_0 в V и вызвать DFS _ Visit _Biconn(v_0) для построения остовного дерева поиска в глубину $S = (V , T)$ и вычисления $v. LOW$ для каждой v в V .

Когда вершина w встречается в DFS _ Visit_Biconn поместить ребро (v, w) в STACK, если его там еще нет.

После обнаружения пары (v, w) такой, что w является сыном v и $w. Low \geq v$, извлечь из STACK все ребра до (v, w) включительно.

Эти ребра образуют двусвязную компоненту графа

(Обратите внимание, что если (v, w) является ребром, то v находится на $Adj [w]$, а w находится на $Adj [v]$. Таким образом, (v, w) встречается дважды: один раз при посещении вершины v *и* один раз при посещении вершины w *.*

Мы можем проверить, находится ли (v, w) уже в STACK, проверив, что $v < w$ и w *является* "старой" или если $v > w$ и $w = v.\pi$.

Пример. Поиск двусвязных компонент

Пример. Дерево поиска в глубину на графе *G* показано с вершинами, переименованными в соответствии с *v.d* и также указаны значения.*Low* .

Например, *DFS _ Visit _Biconn* (6) определяет, что 6. *Low* = 4, поскольку существует обратное ребро (6, 4).

Затем *DFS _ Visit _Biconn* (5), вызвавший *DFS _ Visit _Biconn* (6), устанавливает 5. *Low* = 4, так как 4 меньше начального значения

5.*Low*, которое равно 5.

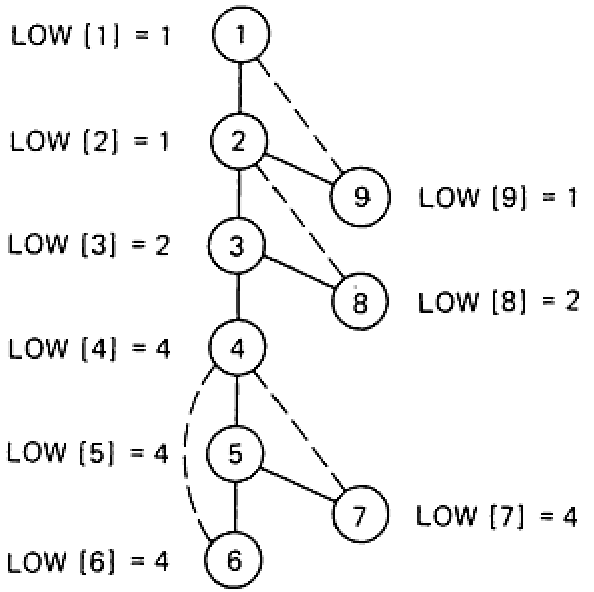
По завершении *DFS _ Visit _Biconn* (5) мы обнаруживаем, что 5.*Low* = 4.

Таким образом, 4 — это точка сочленения.

На этом этапе стек содержит ребра (снизу вверх)

(1, 2) (2, 3), (3, 4), (4, 5), (5, 6), (6, 4). (5, 7), (7, 4).

Таким образом, мы сдвигаем ребра вниз до (4, 5) включительно.



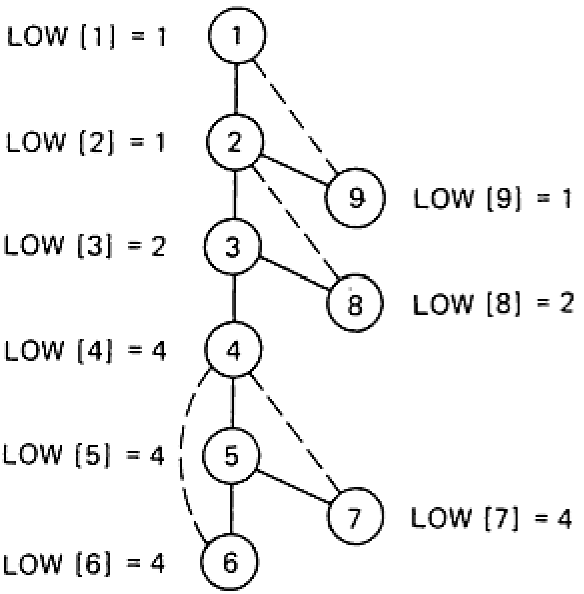
Пример. Поиск двусвязных компонент

То есть мы выводим ребра (7, 4), (5, 7), (6, 4), (5, 6) и (4, 5), которые являются ребрами первой найденной двусвязной компоненты.

Обратите внимание, что по завершении *DFS_visit_Biconn* (2) мы обнаруживаем, что

2.Low = 1 и очищаем стек от ребер даже если 1 не является точкой сочленения.

Это гарантирует, что двусвязная компонента, содержащая корень, будет обнаружена.



- Ваши вопросы?

- Контакты лектора:
  apanovich_09@mail.ru