Обходы ориентированных графов

Владимир Подольский

Факультет компьютерных наук, Высшая Школа Экономики

Обходы ориентированных графов

Обходы ориентированных графов

Поиск компонент сильной связности

Поиск в глубину

- Для неориентированных графов у нас были поиск в глубину и поиск в ширину

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```

Поиск в глубину

- Для неориентированных графов у нас были поиск в глубину и поиск в ширину
- Буквально те же алгоритмы работают и для ориентированных графов!

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```

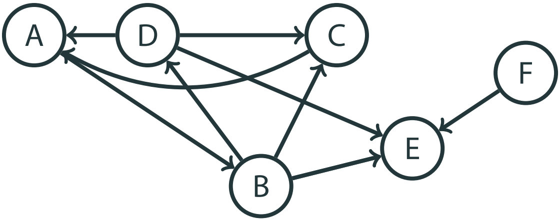
Поиск в глубину

- Для неориентированных графов у нас были поиск в глубину и поиск в ширину
- Буквально те же алгоритмы работают и для ориентированных графов!
- Посмотрим на примере поиска в глубину

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```

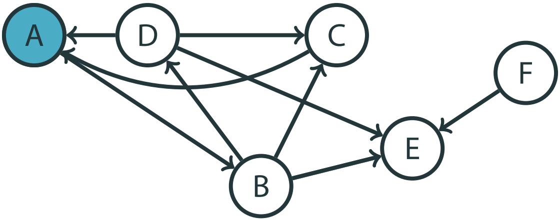
Поиск в глубину

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```



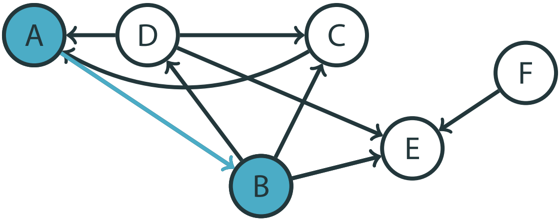
Поиск в глубину

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```



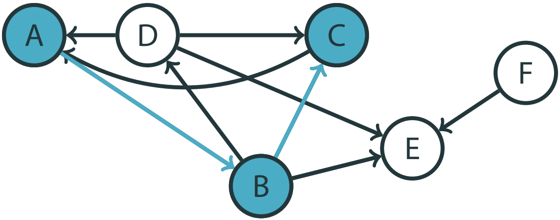
Поиск в глубину

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```



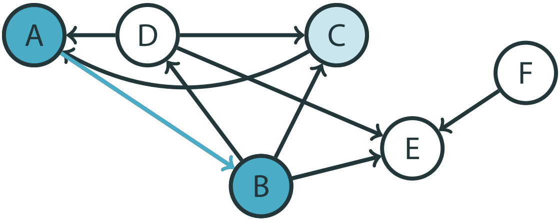
Поиск в глубину

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```



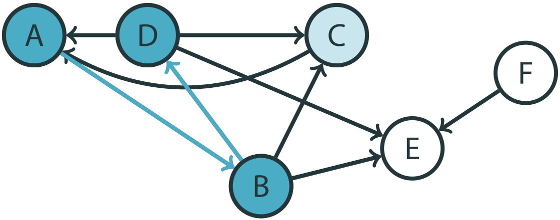
Поиск в глубину

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```



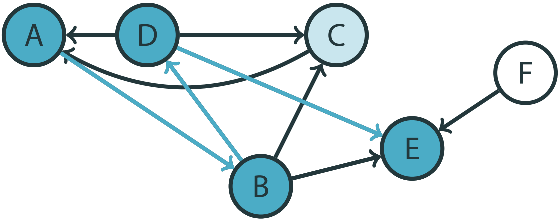
Поиск в глубину

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```



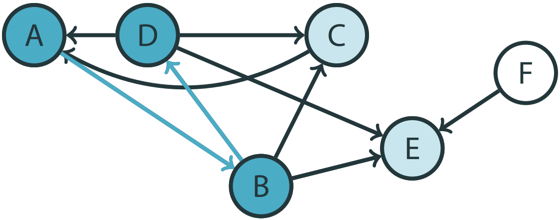
Поиск в глубину

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```



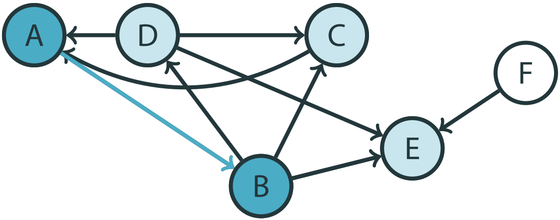
Поиск в глубину

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```



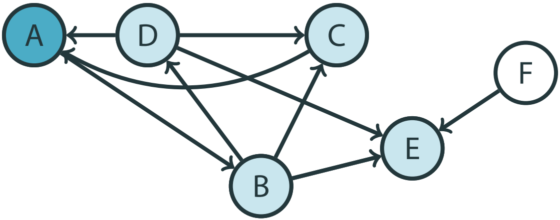
Поиск в глубину

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```



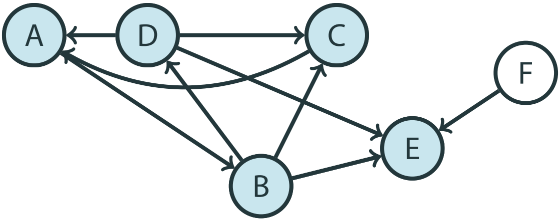
Поиск в глубину

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```



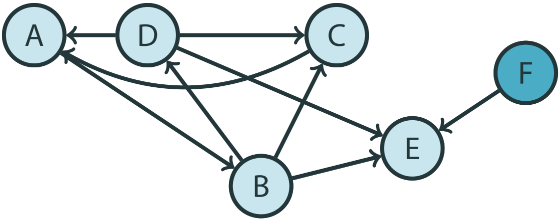
Поиск в глубину

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```



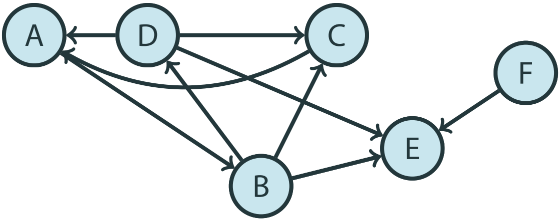
Поиск в глубину

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```



Поиск в глубину

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```



Время обработки

- Напомним, что можно считать время обработки вершин

```
def Previsit(v):
    pre[v]=clock
    clock+=1

def Postvisit(v):
    post[v]=clock
    clock+=1
```

Время обработки

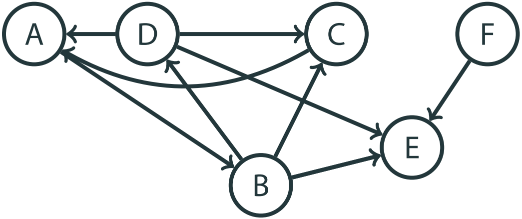
- Напомним, что можно считать время обработки вершин
- Для любой вершины v у нас два числа: $pre[v]$ и $post[v]$

```
def Previsit(v):
    pre[v]=clock
    clock+=1

def Postvisit(v):
    post[v]=clock
    clock+=1
```

Поиск в глубину

count=0

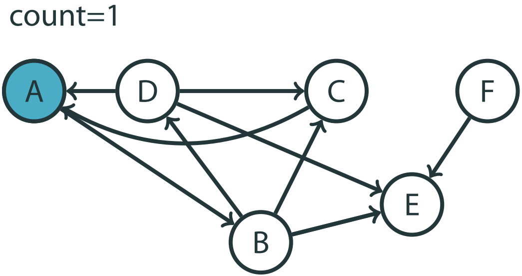



```
def  Previsit(v):
     pre[v]=clock
     clock+=1

def  Postvisit(v):
     post[v]=clock
     clock+=1
```

	pre	post
A		
B		
C		
D		
E		
F		

Поиск в глубину

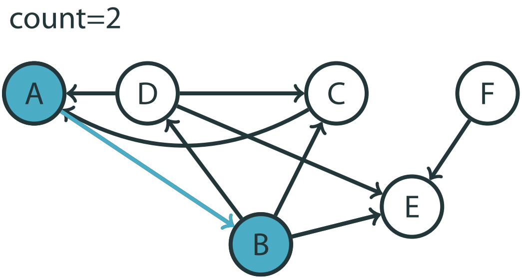


	pre	post
A	0	
B		
C		
D		
E		
F		

```
def Previsit(v):
    pre[v]=clock
    clock+=1

def Postvisit(v):
    post[v]=clock
    clock+=1
```

Поиск в глубину

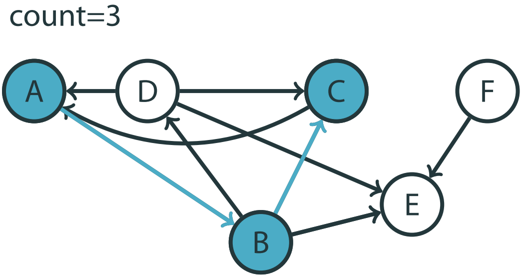



```
def Previsit(v):
    pre[v]=clock
    clock+=1

def Postvisit(v):
    post[v]=clock
    clock+=1
```

	pre	post
A	0	
B	1	
C		
D		
E		
F		

Поиск в глубину

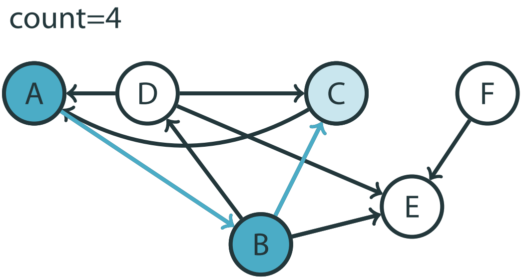


	pre	post
A	0	
B	1	
C	2	
D		
E		
F		

```
def Previsit(v):
    pre[v]=clock
    clock+=1

def Postvisit(v):
    post[v]=clock
    clock+=1
```

Поиск в глубину

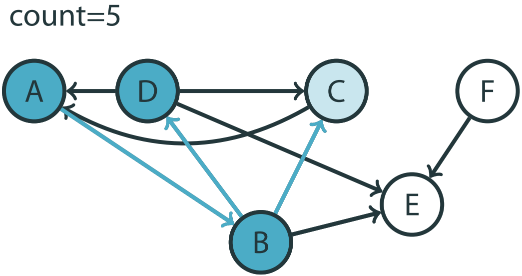



```
def  Previsit(v):
     pre[v]=clock
     clock+=1

def  Postvisit(v):
     post[v]=clock
     clock+=1
```

	pre	post
A	0	
B	1	
C	2	3
D		
E		
F		

Поиск в глубину

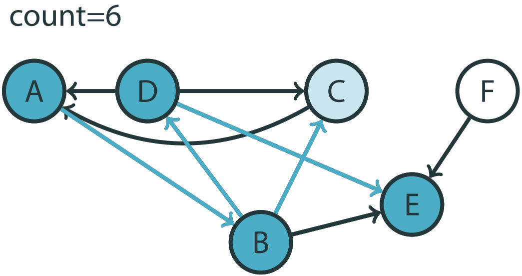



```
def Previsit(v):
    pre[v]=clock
    clock+=1

def Postvisit(v):
    post[v]=clock
    clock+=1
```

	pre	post
A	0	
B	1	
C	2	3
D	4	
E		
F		

Поиск в глубину

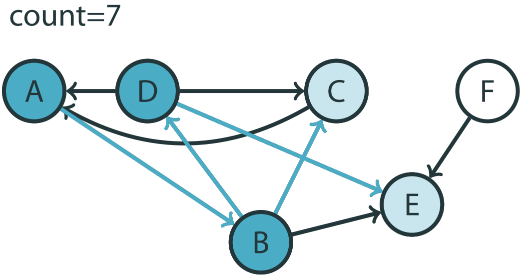


	pre	post
A	0	
B	1	
C	2	3
D	4	
E	5	
F		

```
def  Previsit(v):
     pre[v]=clock
     clock+=1

def  Postvisit(v):
     post[v]=clock
     clock+=1
```

Поиск в глубину

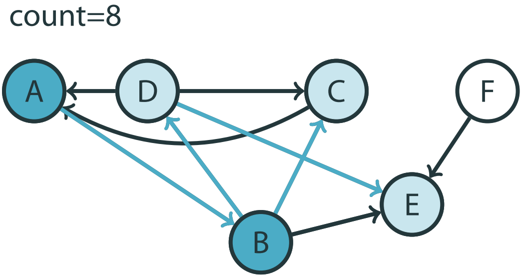



```
def Previsit(v):
    pre[v]=clock
    clock+=1

def Postvisit(v):
    post[v]=clock
    clock+=1
```

	pre	post
A	0	
B	1	
C	2	3
D	4	
E	5	6
F		

Поиск в глубину

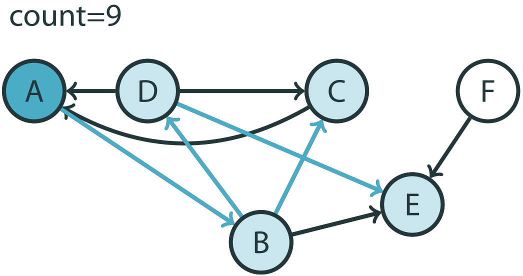



```
def Previsit(v):
    pre[v]=clock
    clock+=1

def Postvisit(v):
    post[v]=clock
    clock+=1
```

	pre	post
A	0	
B	1	
C	2	3
D	4	7
E	5	6
F		

Поиск в глубину

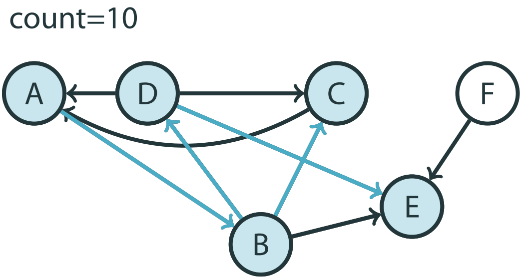



```
def  Previsit(v):
     pre[v]=clock
     clock+=1

def  Postvisit(v):
     post[v]=clock
     clock+=1
```

	pre	post
A	0	
B	1	8
C	2	3
D	4	7
E	5	6
F		

Поиск в глубину

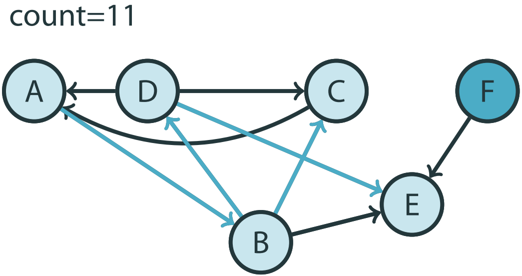


	pre	post
A	0	9
B	1	8
C	2	3
D	4	7
E	5	6
F		

```
def  Previsit(v):
     pre[v]=clock
     clock+=1

def  Postvisit(v):
     post[v]=clock
     clock+=1
```

Поиск в глубину

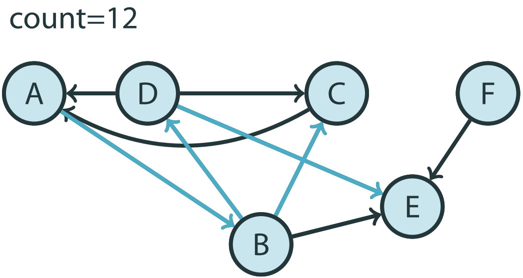



```
def  Previsit(v):
     pre[v]=clock
     clock+=1

def  Postvisit(v):
     post[v]=clock
     clock+=1
```

	pre	post
A	0	9
B	1	8
C	2	3
D	4	7
E	5	6
F	10	

Поиск в глубину

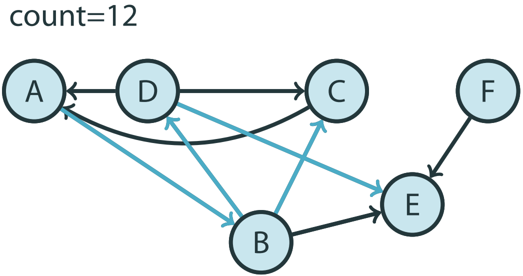



```
def  Previsit(v):
     pre[v]=clock
     clock+=1

def  Postvisit(v):
     post[v]=clock
     clock+=1
```

	pre	post
A	0	9
B	1	8
C	2	3
D	4	7
E	5	6
F	10	11

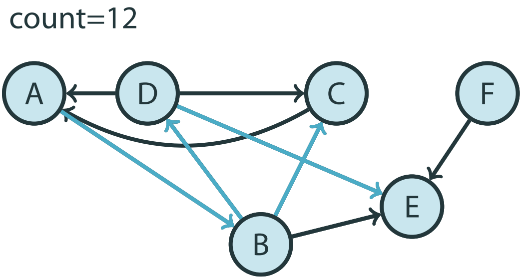
Поиск в глубину



	pre	post
A	0	9
B	1	8
C	2	3
D	4	7
E	5	6
F	10	11

- 4 типа ребер

Поиск в глубину

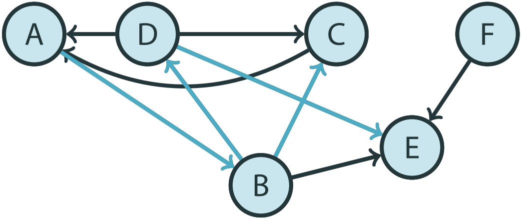


	pre	post
A	0	9
B	1	8
C	2	3
D	4	7
E	5	6
F	10	11

- 4 типа ребер
- Древесные — ребра в обходе

Поиск в глубину

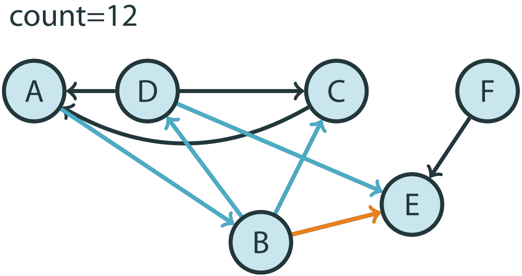
count=12



	pre	post
A	0	9
B	1	8
C	2	3
D	4	7
E	5	6
F	10	11

- 4 типа ребер
- Древесные — ребра в обходе
- Отрезки [pre,post] вложены, post в конце меньше

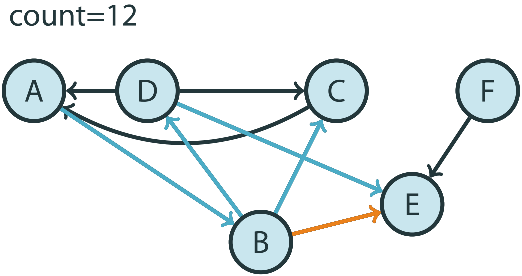
Поиск в глубину



	pre	post
A	0	9
B	1	8
C	2	3
D	4	7
E	5	6
F	10	11

- Прямые — ребра из вершин в их последователя в дереве

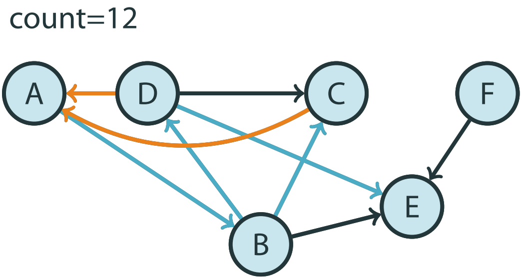
Поиск в глубину



	pre	post
A	0	9
B	1	8
C	2	3
D	4	7
E	5	6
F	10	11

- Прямые — ребра из вершин в их последователя в дереве
- Отрезки [pre,post] вложены, post в конце меньше

Поиск в глубину

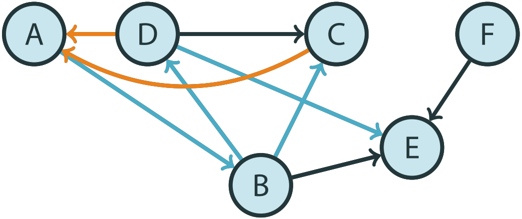


	pre	post
A	0	9
B	1	8
C	2	3
D	4	7
E	5	6
F	10	11

- Обратные — ребра из вершин в их предшественника в дереве

Поиск в глубину

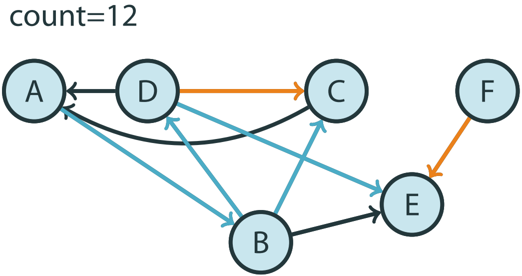
count=12



- Обратные — ребра из вершин в их предшественника в дереве
- Отрезки [pre,post] вложены, post в конце больше!

	pre	post
A	0	9
B	1	8
C	2	3
D	4	7
E	5	6
F	10	11

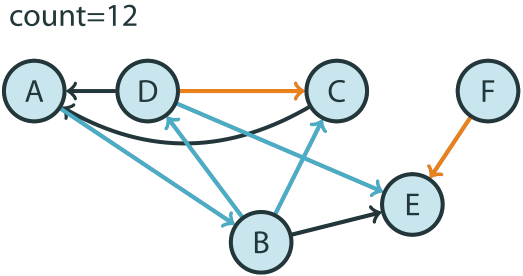
Поиск в глубину



- Перекрестные — ребра, несравнимые в деревьях

	pre	post
A	0	9
B	1	8
C	2	3
D	4	7
E	5	6
F	10	11

Поиск в глубину



	pre	post
A	0	9
B	1	8
C	2	3
D	4	7
E	5	6
F	10	11

- Перекрестные — ребра, несравнимые в деревьях
- Отрезки [pre,post] не пересекаются, post в конце меньше

Проверка на ацикличность

- Легко проверять, есть ли обратные ребра: просто проверяем, увеличивается ли post вдоль ребер

Проверка на ацикличность

- Легко проверять, есть ли обратные ребра: просто проверяем, увеличивается ли post вдоль ребер
- Это позволяет проверять граф на ацикличность

Проверка на ацикличность

- Легко проверять, есть ли обратные ребра: просто проверяем, увеличивается ли post вдоль ребер
- Это позволяет проверять граф на ацикличность
- Цикл есть тогда и только тогда, когда есть обратные ребра

Проверка на ацикличность

- Легко проверять, есть ли обратные ребра: просто проверяем, увеличивается ли post вдоль ребер
- Это позволяет проверять граф на ацикличность
- Цикл есть тогда и только тогда, когда есть обратные ребра
- Действительно, если есть обратные ребра, есть цикл

Проверка на ацикличность

- Легко проверять, есть ли обратные ребра: просто проверяем, увеличивается ли post вдоль ребер
- Это позволяет проверять граф на ацикличность
- Цикл есть тогда и только тогда, когда есть обратные ребра
- Действительно, если есть обратные ребра, есть цикл
- Если есть цикл, посмотрим на первую вершину цикла, которая попала в обход

Проверка на ацикличность

- Легко проверять, есть ли обратные ребра: просто проверяем, увеличивается ли post вдоль ребер
- Это позволяет проверять граф на ацикличность
- Цикл есть тогда и только тогда, когда есть обратные ребра
- Действительно, если есть обратные ребра, есть цикл
- Если есть цикл, посмотрим на первую вершину цикла, которая попала в обход
- Остальные вершины цикла будут среди ее последователей

Проверка на ацикличность

- Легко проверять, есть ли обратные ребра: просто проверяем, увеличивается ли post вдоль ребер
- Это позволяет проверять граф на ацикличность
- Цикл есть тогда и только тогда, когда есть обратные ребра
- Действительно, если есть обратные ребра, есть цикл
- Если есть цикл, посмотрим на первую вершину цикла, которая попала в обход
- Остальные вершины цикла будут среди ее последователей
- Будет обратное ребро

Обходы ориентированных графов

Обходы ориентированных графов

Поиск компонент сильной связности

Компоненты сильной связности

- Как искать компоненты сильной связности?

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```

Компоненты сильной связности

- Как искать компоненты сильной связности?
- Для этого можно использовать поиск в глубину!

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```

Компоненты сильной связности

- Как искать компоненты сильной связности?
- Для этого можно использовать поиск в глубину!
- Что будет если запустить процедуру Explore в вершине v?

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```

Компоненты сильной связности

- Как искать компоненты сильной связности?
- Для этого можно использовать поиск в глубину!
- Что будет если запустить процедуру Explore в вершине v?
- Она обойдет все вершины, достижимые из v, и остановится

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```

Компоненты сильной связности

- Идея: запустить Explore из вершины компоненты-стока

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```

Компоненты сильной связности

- Идея: запустить Explore из вершины компоненты-стока
- Тогда обойдем только вершины компоненты стока

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```

Компоненты сильной связности

- Идея: запустить Explore из вершины компоненты-стока
- Тогда обойдем только вершины компоненты стока
- Удалим вершины этой компоненты и повторим

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```

Компоненты сильной связности

- Идея: запустить Explore из вершины компоненты-стока
- Тогда обойдем только вершины компоненты стока
- Удалим вершины этой компоненты и повторим
- Проблема: как найти вершину компоненты-стока?

```
def Explore(v):
    visited[v]=True
    Previsit(v)
    for u in graph[v]:
        if not visited[u]:
            Explore(u)
    Postvisit(v)
```

Компоненты сильной связности

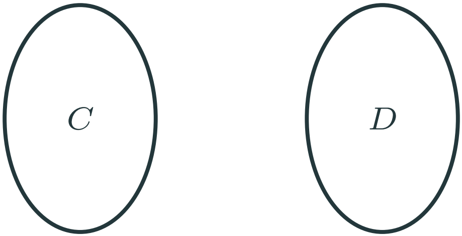
- Сделаем следующее наблюдение

Компоненты сильной связности

- Сделаем следующее наблюдение
- Запустим поиск в глубину со счетчиками времени

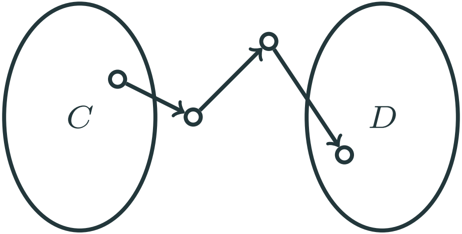
Компоненты сильной связности

- Пусть C и D — две компоненты сильной связности



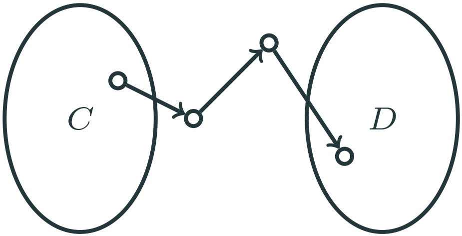
Компоненты сильной связности

- Пусть C и D — две компоненты сильной связности
- Пусть есть путь из какой-то вершины C в какую-то вершину D



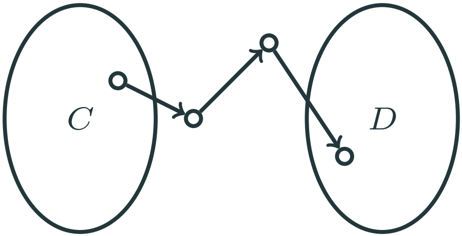
Компоненты сильной связности

- Рассмотрим в каждой компоненте по вершине с максимальным post



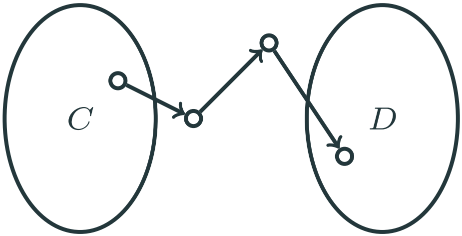
Компоненты сильной связности

- Рассмотрим в каждой компоненте по вершине с максимальным post
- Тогда в компоненте C эта величина post больше!



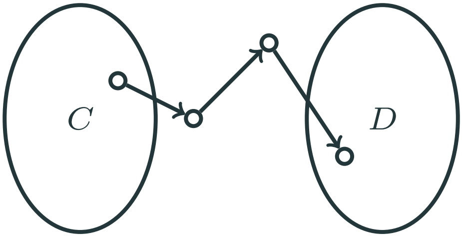
Компоненты сильной связности

- Почему это так?



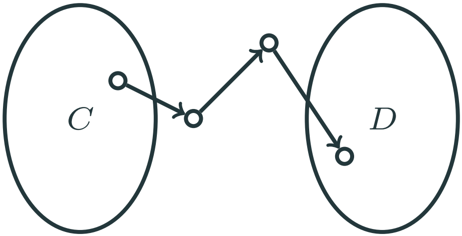
Компоненты сильной связности

- Почему это так?
- Посмотрим на вершину из C или D, которая встретиться первой в обходе



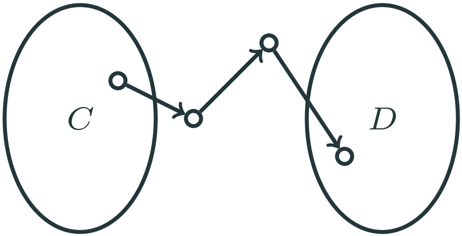
Компоненты сильной связности

- Если она из C, то ее обход обойдет все вершины из D



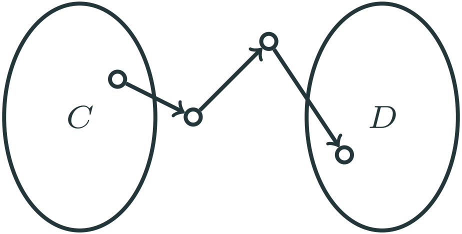
Компоненты сильной связности

- Если она из C, то ее обход обойдет все вершины из D
- И ее post будет больше, чем у всех вершин из D



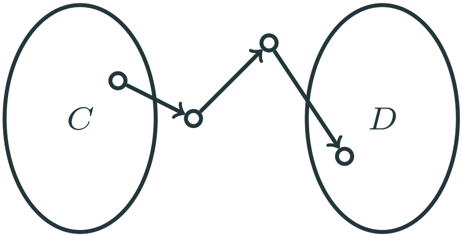
Компоненты сильной связности

- Если она из D, то ее обход обойдет все вершины из D



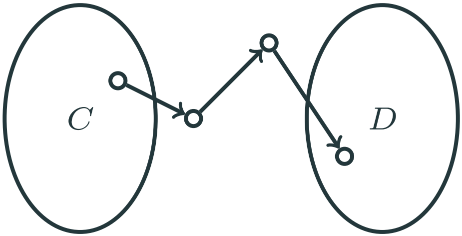
Компоненты сильной связности

- Если она из D, то ее обход обойдет все вершины из D
- Но не вершины из C!

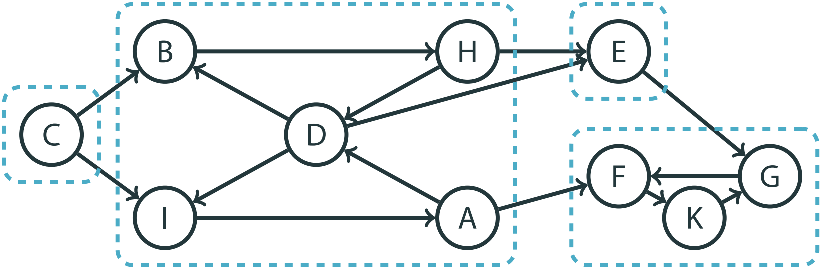


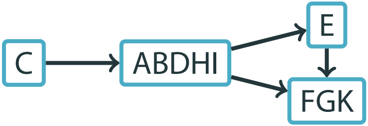
Компоненты сильной связности

- Если она из D, то ее обход обойдет все вершины из D
- Но не вершины из C!
- Они будут обойдены позже, и у них будет больше post



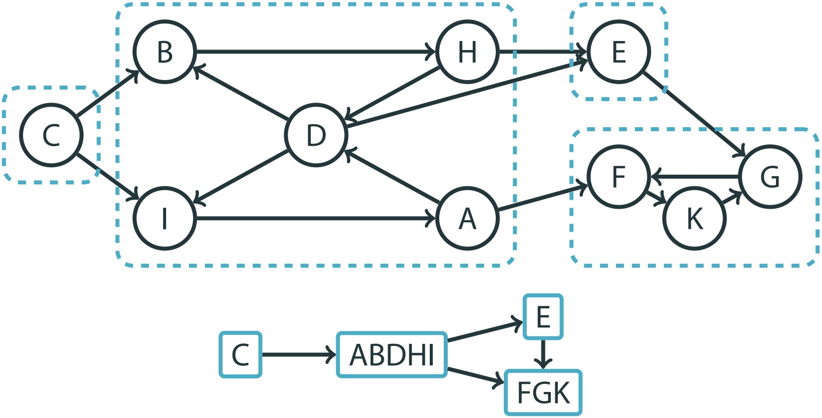
Компоненты сильной связности





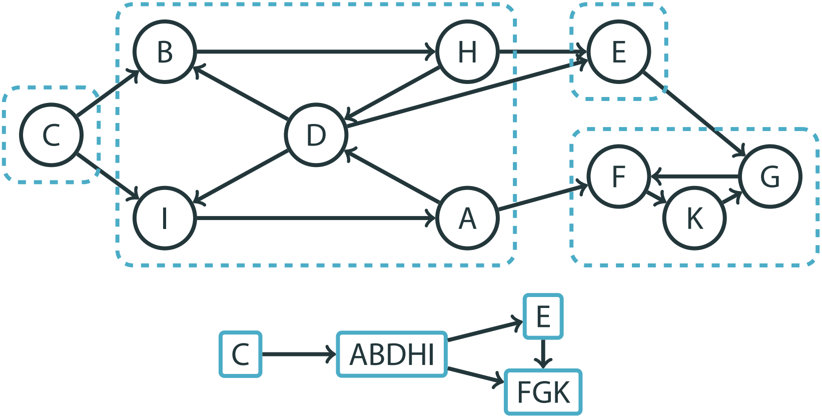
- Максимальное значение post вершин компоненты тем больше, чем раньше лежит компонента в графе компонент

Компоненты сильной связности



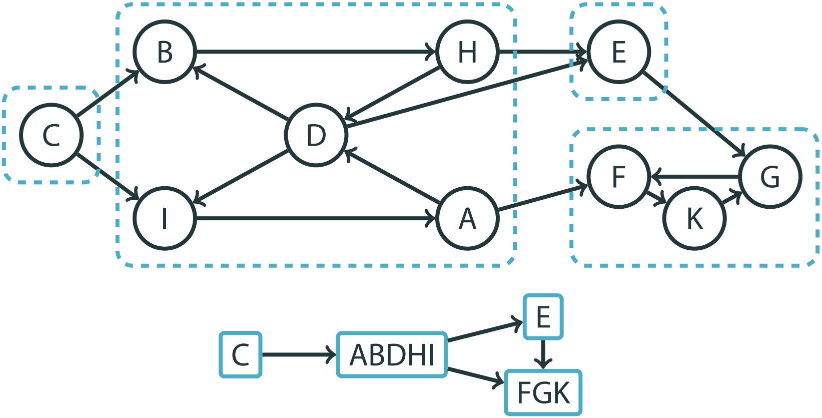
- Где лежит вершина с самым большим post?

Компоненты сильной связности



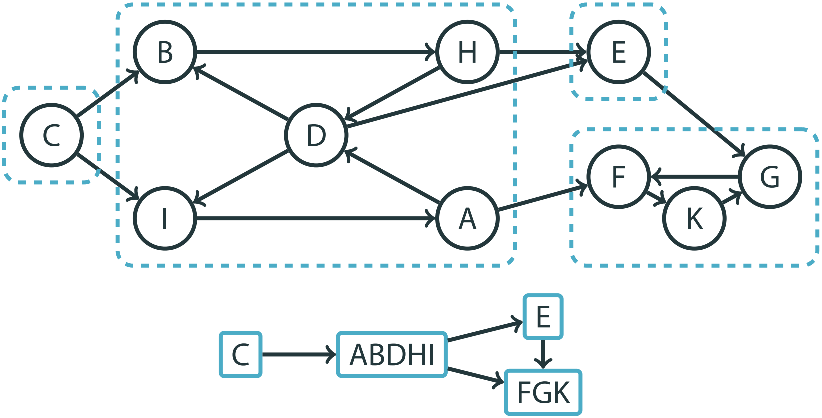
- Где лежит вершина с самым большим post?
- В компоненте-истоке!

Компоненты сильной связности



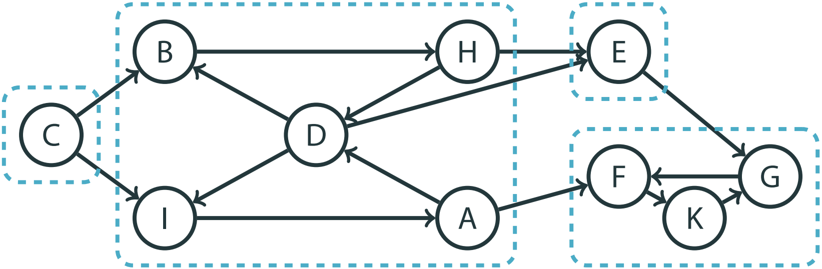
- Можем найти вершину в компоненте-истоке с помощью поиска в глубину

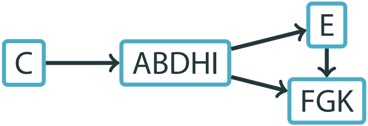
Компоненты сильной связности



- Но нам нужна была вершина в компоненте-стоке

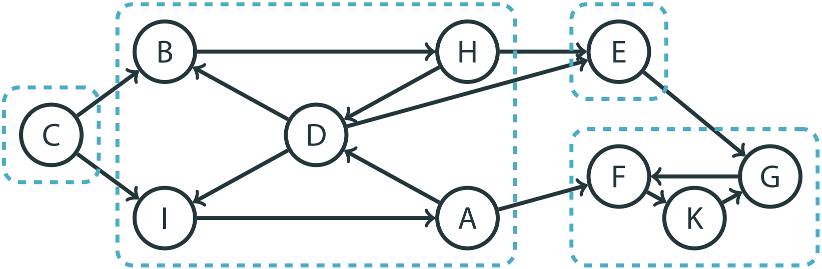
Компоненты сильной связности

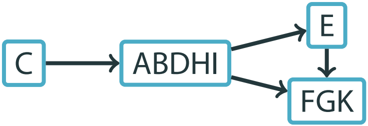




- Но нам нужна была вершина в компоненте-стоке
- Как же это поправить?

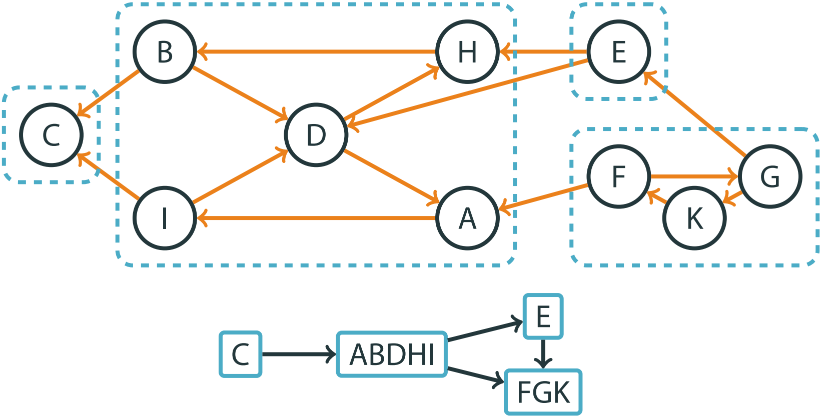
Компоненты сильной связности





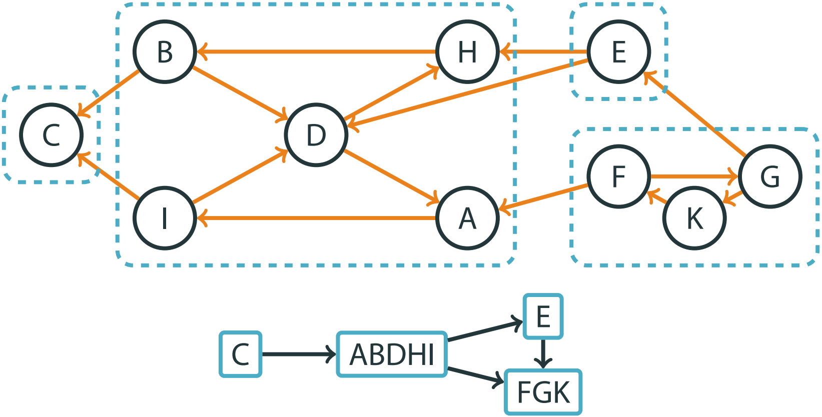
- Идея: развернем все ребра в графе

Компоненты сильной связности



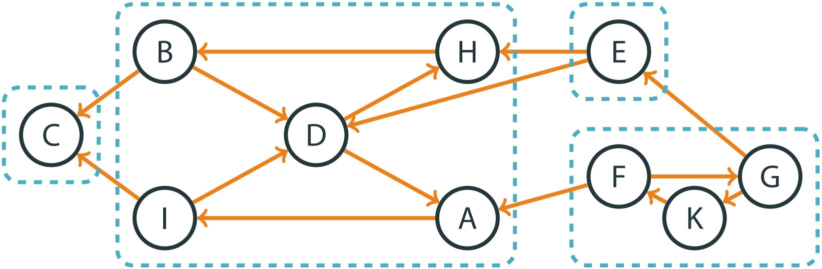
- Идея: развернем все ребра в графе

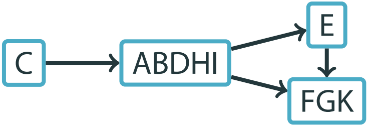
Компоненты сильной связности



- Идея: развернем все ребра в графе
- Что станет с компонентами связности?

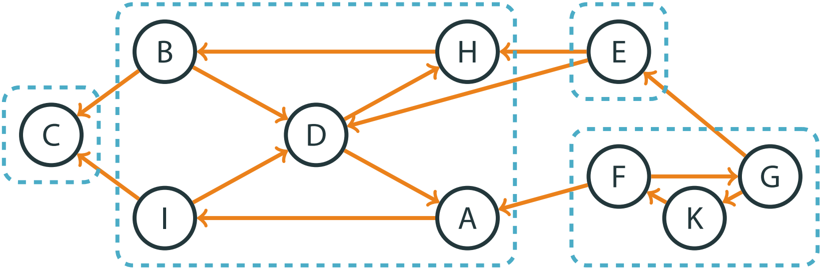
Компоненты сильной связности

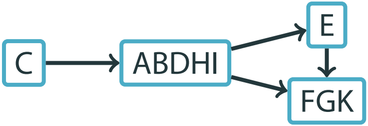




- Они останутся те же!

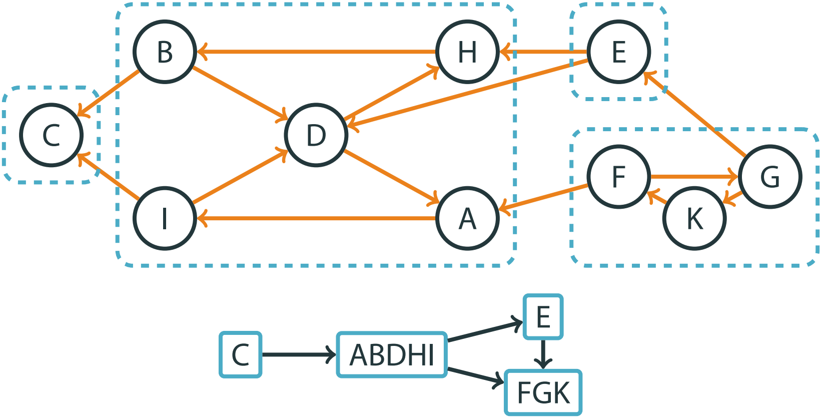
Компоненты сильной связности





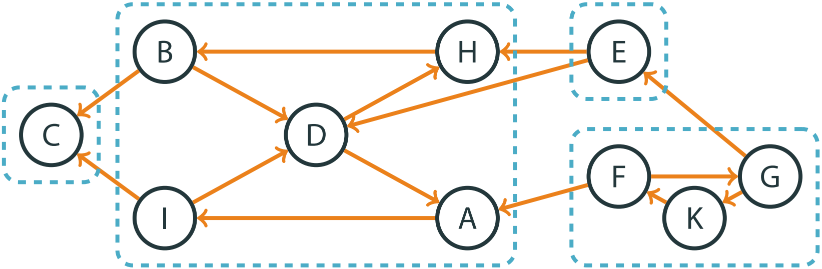
- Они останутся те же!
- Что станет с графом компонент?

Компоненты сильной связности



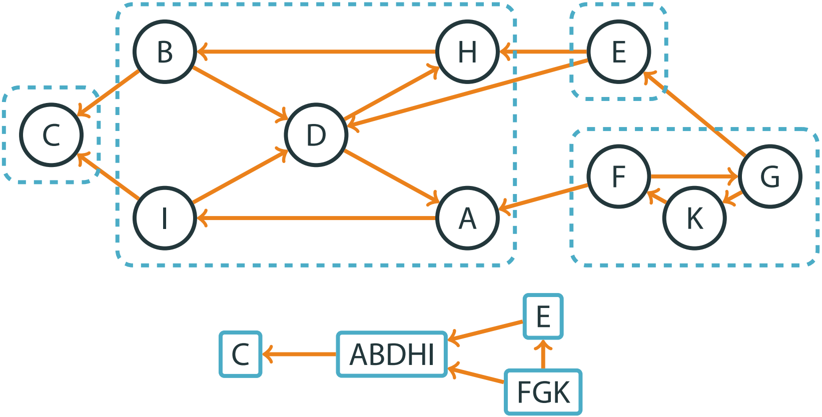
- Ребра в нем развернутся

Компоненты сильной связности



- Ребра в нем развернутся

Компоненты сильной связности



- Ребра в нем развернутся
- Исток и сток поменяются местами!

Общая схема

- Пусть дан граф G

Общая схема

- Пусть дан граф G
- Рассмотрим граф G^R с развернутыми ребрами

Общая схема

- Пусть дан граф G
- Рассмотрим граф G^R с развернутыми ребрами
- Запустим поиск в глубину в графе G^R

Общая схема

- Пусть дан граф G
- Рассмотрим граф G^R с развернутыми ребрами
- Запустим поиск в глубину в графе G^R
- Рассмотрим вершину v с максимальным post

Общая схема

- Пусть дан граф G
- Рассмотрим граф G^R с развернутыми ребрами
- Запустим поиск в глубину в графе G^R
- Рассмотрим вершину v с максимальным post
- Она лежит в компоненте-стоке графа G

Общая схема

- Пусть дан граф G
- Рассмотрим граф G^R с развернутыми ребрами
- Запустим поиск в глубину в графе G^R
- Рассмотрим вершину v с максимальным post
- Она лежит в компоненте-стоке графа G
- Запустим Explore(v) в графе G

Общая схема

- Пусть дан граф G
- Рассмотрим граф G^R с развернутыми ребрами
- Запустим поиск в глубину в графе G^R
- Рассмотрим вершину v с максимальным post
- Она лежит в компоненте-стоке графа G
- Запустим Explore(v) в графе G
- Он найдет компоненту-сток

Общая схема

- Пусть дан граф G
- Рассмотрим граф G^R с развернутыми ребрами
- Запустим поиск в глубину в графе G^R
- Рассмотрим вершину v с максимальным post
- Она лежит в компоненте-стоке графа G
- Запустим Explore(v) в графе G
- Он найдет компоненту-сток
- Удалим ее и повторим

Общая схема

- Что именно нужно повторять?

Общая схема

- Что именно нужно повторять?
- У нас два шага: найти вершину и запустить Explore из нее

Общая схема

- Что именно нужно повторять?
- У нас два шага: найти вершину и запустить Explore из нее
- После удаления компоненты-стока мы знаем вершину в новом стоке G!

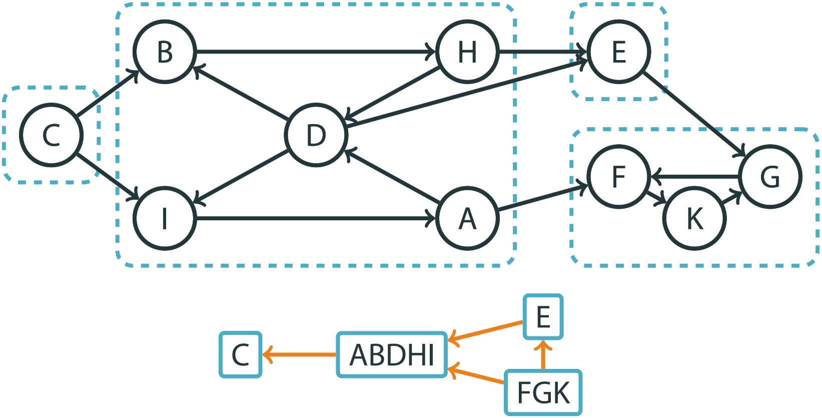
Общая схема

- Что именно нужно повторять?
- У нас два шага: найти вершину и запустить Explore из нее
- После удаления компоненты-стока мы знаем вершину в новом стоке G!
- У нее максимальный post из оставшихся (при обходе G^R)

Общая схема

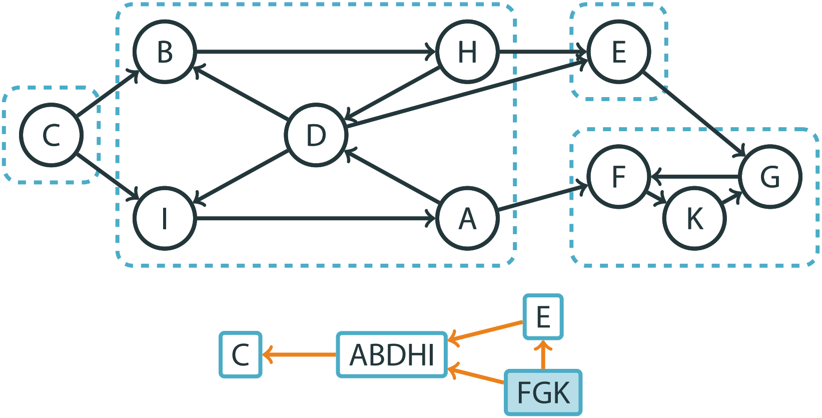
- Что именно нужно повторять?
- У нас два шага: найти вершину и запустить Explore из нее
- После удаления компоненты-стока мы знаем вершину в новом стоке G!
- У нее максимальный post из оставшихся (при обходе G^R)
- Так что повторять достаточно лишь запуск Explore(v) из новых вершин v

Пример



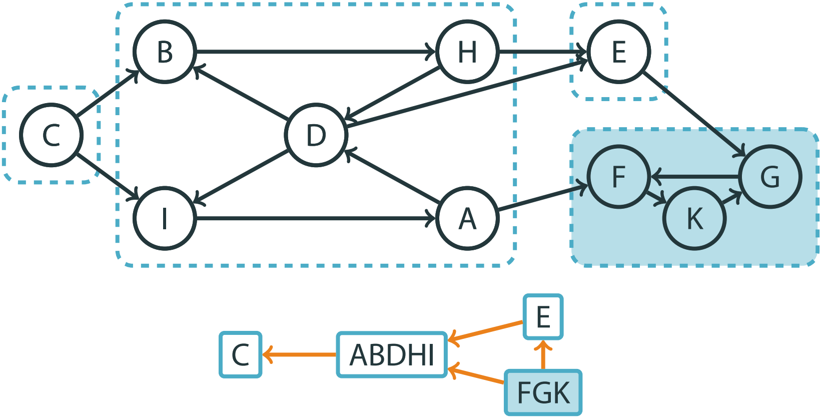
- Запустим поиск в глубину в графе G^R

Пример



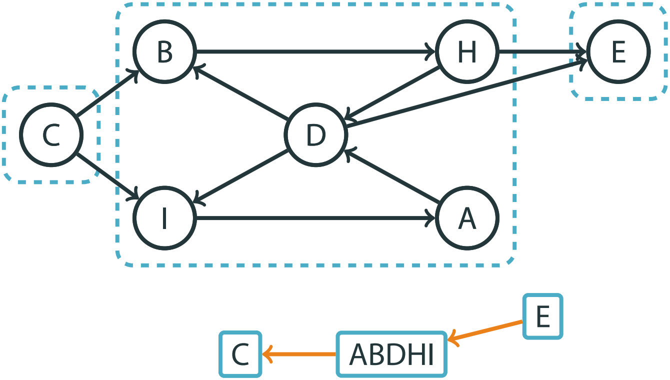
- Запустим поиск в глубину в графе G^R
- Возьмем вершину с максимальным post

Пример



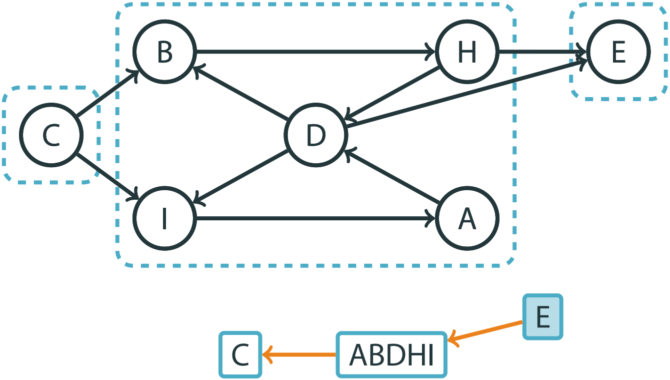
- Запустим поиск в глубину

Пример



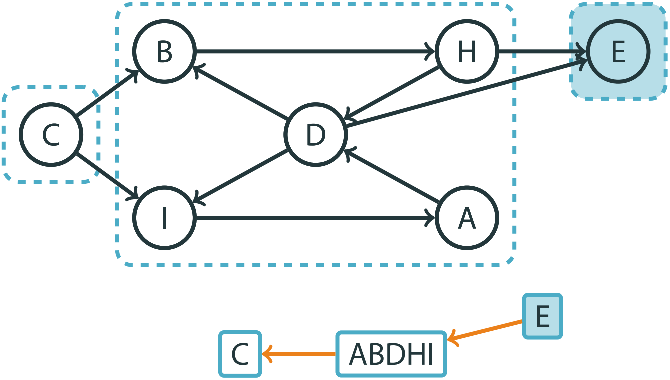
- Запустим поиск в глубину
- Удалим компоненту FGK

Пример



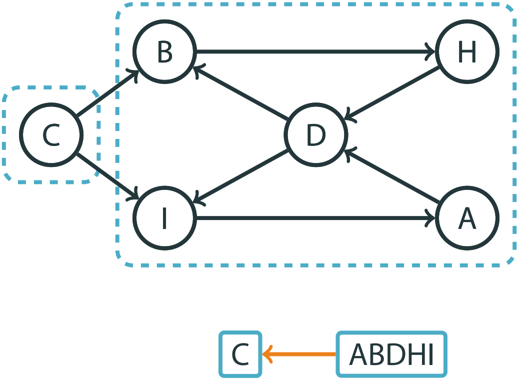
- Возьмем вершину с максимальным post

Пример



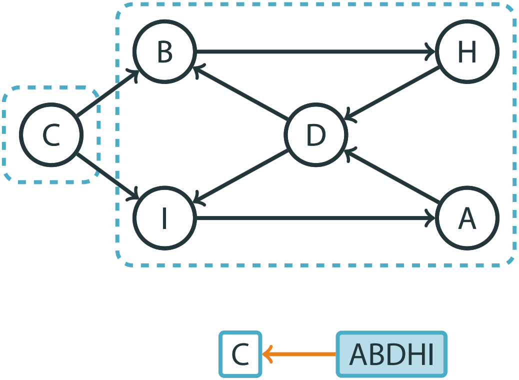
- Запустим поиск

Пример



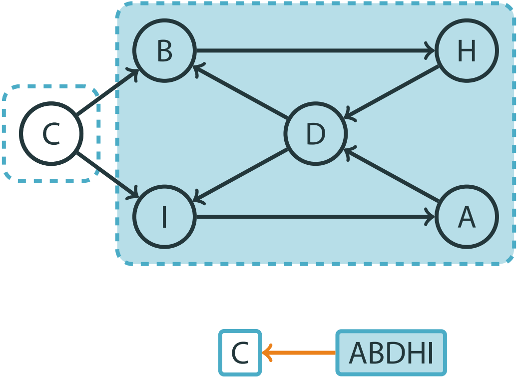
- Запустим поиск
- Удалим компоненту E

Пример



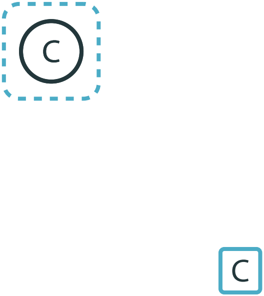
- Возьмем вершину с максимальным post

Пример



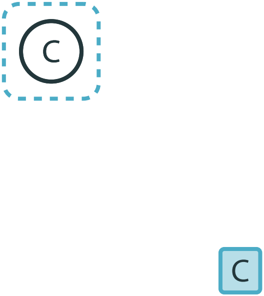
- Запустим поиск

Пример



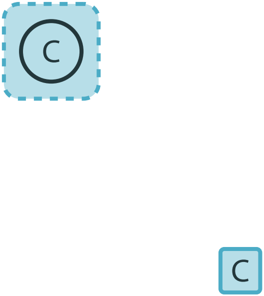
- Запустим поиск
- Удалим компоненту ABDHI

Пример



- Возьмем вершину с максимальным post

Пример



- Запустим поиск

Пример

- Запустим поиск
- Удалим компоненту C

Эффективность

- Оценим, насколько быстро это работает

Эффективность

- Оценим, насколько быстро это работает
- По сути мы запускаем поиск глубину в G^R

Эффективность

- Оценим, насколько быстро это работает
- По сути мы запускаем поиск глубину в G^R
- А затем поиск в глубину в G

Эффективность

- Оценим, насколько быстро это работает
- По сути мы запускаем поиск глубину в G^R
- А затем поиск в глубину в G
- Время работы примерно как у поиска в глубину

Что мы узнали

- На этой неделе мы обсудили важный класс графов — деревья

Что мы узнали

- На этой неделе мы обсудили важный класс графов — деревья
- Также мы обсудили ориентированные графы

Что мы узнали

- На этой неделе мы обсудили важный класс графов — деревья
- Также мы обсудили ориентированные графы
- Вопросы о достижимости и связности для них сложнее

Что мы узнали

- На этой неделе мы обсудили важный класс графов — деревья
- Также мы обсудили ориентированные графы
- Вопросы о достижимости и связности для них сложнее
- Поиск в глубину оказывается очень полезным

Что мы узнали

- На этой неделе мы обсудили важный класс графов — деревья
- Также мы обсудили ориентированные графы
- Вопросы о достижимости и связности для них сложнее
- Поиск в глубину оказывается очень полезным
- На следующей неделе обсудим случайные блуждания в графах и их приложения