Поиск в глубину

Владимир Подольский

Факультет компьютерных наук, Высшая Школа Экономики

Пути и достижимость

Обходы и поиск в графах

Обходы в графах

- В большом числе задач на графах требуется обойти вершины графа, проходя при этом по ребрам

Обходы в графах

- В большом числе задач на графах требуется обойти вершины графа, проходя при этом по ребрам
- Например, проверка связности

Обходы в графах

- В большом числе задач на графах требуется обойти вершины графа, проходя при этом по ребрам
- Например, проверка связности
- Как же делать это эффективно?

Обходы в графах

- В большом числе задач на графах требуется обойти вершины графа, проходя при этом по ребрам
- Например, проверка связности
- Как же делать это эффективно?
- Оказывается, что очень полезными оказываются рекурсивные обходы

Поиск в глубину

- Будем помечать посещенные вершины

Поиск в глубину

- Будем помечать посещенные вершины
- Стартуем с какой-то вершины, помечаем ее как посещенную

Поиск в глубину

- Будем помечать посещенные вершины
- Стартуем с какой-то вершины, помечаем ее как посещенную
- Находясь в очередной вершине храним текущий пройденный путь из начальной вершины

Поиск в глубину

- Перебираем соседей по очереди, пока не найдем не посещенную вершину

Поиск в глубину

- Перебираем соседей по очереди, пока не найдем не посещенную вершину
- Если нашли, переходим в нее, помечаем ее как посещенную, добавляем ее в пройденный путь

Поиск в глубину

- Перебираем соседей по очереди, пока не найдем не посещенную вершину
- Если нашли, переходим в нее, помечаем ее как посещенную, добавляем ее в пройденный путь
- Если не нашли, возвращаемся на одну вершину назад, в ней продолжаем искать непосещенного соседа (пройденный путь укорачивается на 1)

Поиск в глубину

- Перебираем соседей по очереди, пока не найдем не посещенную вершину
- Если нашли, переходим в нее, помечаем ее как посещенную, добавляем ее в пройденный путь
- Если не нашли, возвращаемся на одну вершину назад, в ней продолжаем искать непосещенного соседа (пройденный путь укорачивается на 1)
- Когда вернемся в изначальную вершину, обход заканчивается

Поиск в глубину

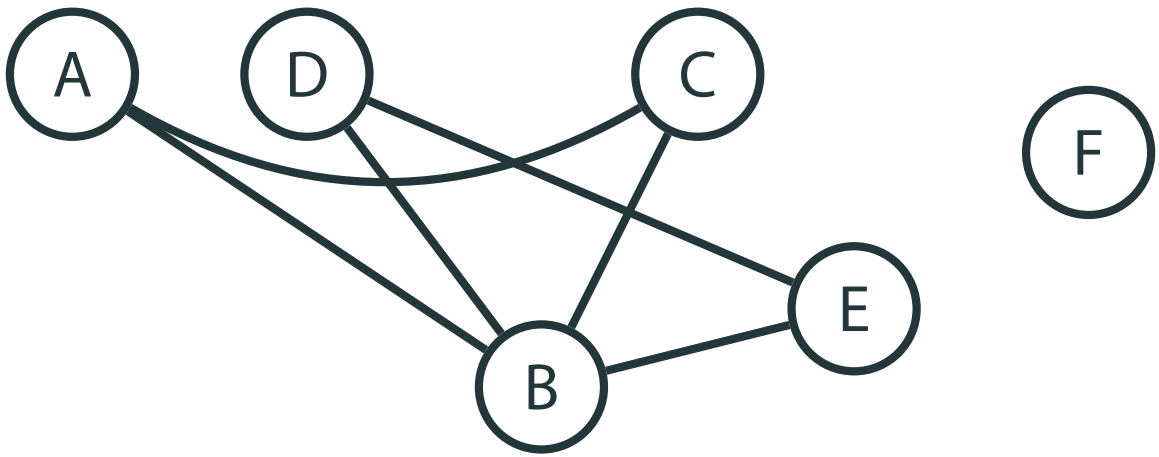
- Это можно описать рекурсивно

```
def Explore(v):
    visited[v]=True

    for u in graph[v]:
        if not visited[u]:
            Explore(u)
```

Поиск в глубину

- Это можно описать рекурсивно

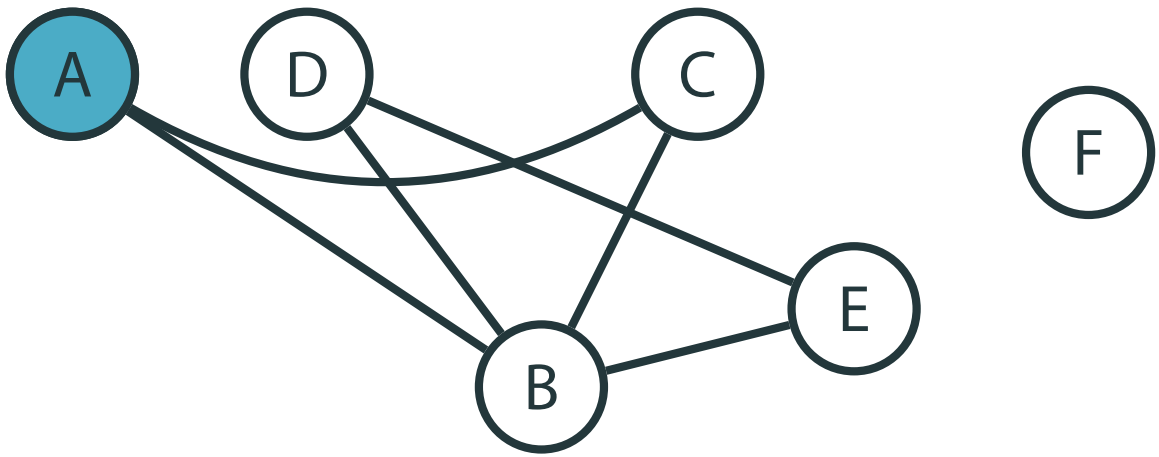
```
def Explore(v):
    visited[v]=True

    for u in graph[v]:
        if not visited[u]:
            Explore(u)
```



Поиск в глубину

- Это можно описать рекурсивно

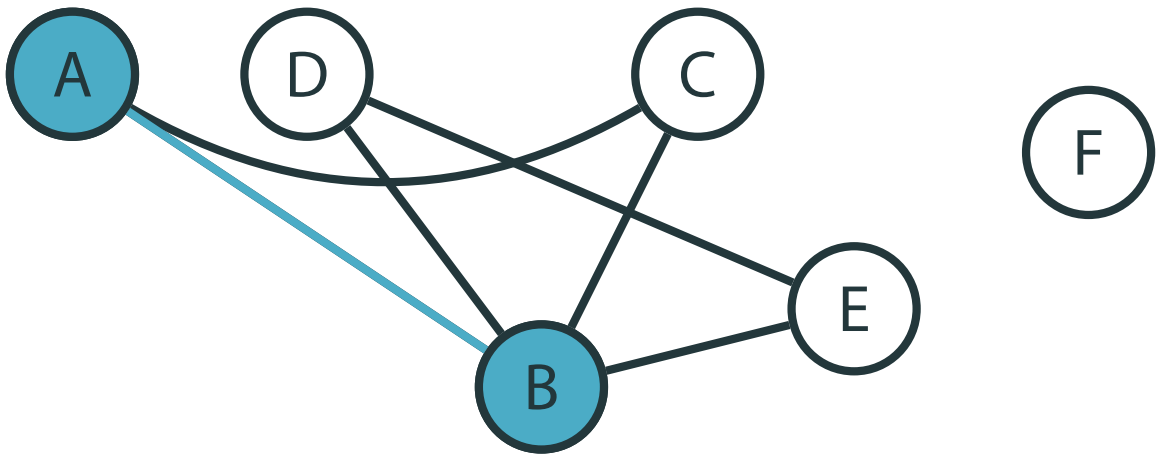
```
def Explore(v):
    visited[v]=True

    for u in graph[v]:
        if not visited[u]:
            Explore(u)
```



Поиск в глубину

- Это можно описать рекурсивно

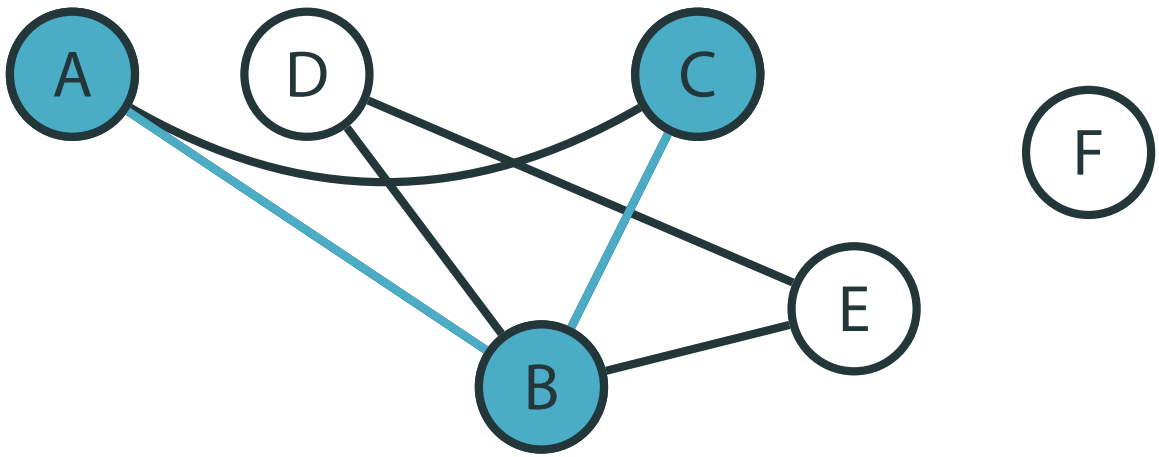
```
def Explore(v):
    visited[v]=True

    for u in graph[v]:
        if not visited[u]:
            Explore(u)
```



Поиск в глубину

- Это можно описать рекурсивно

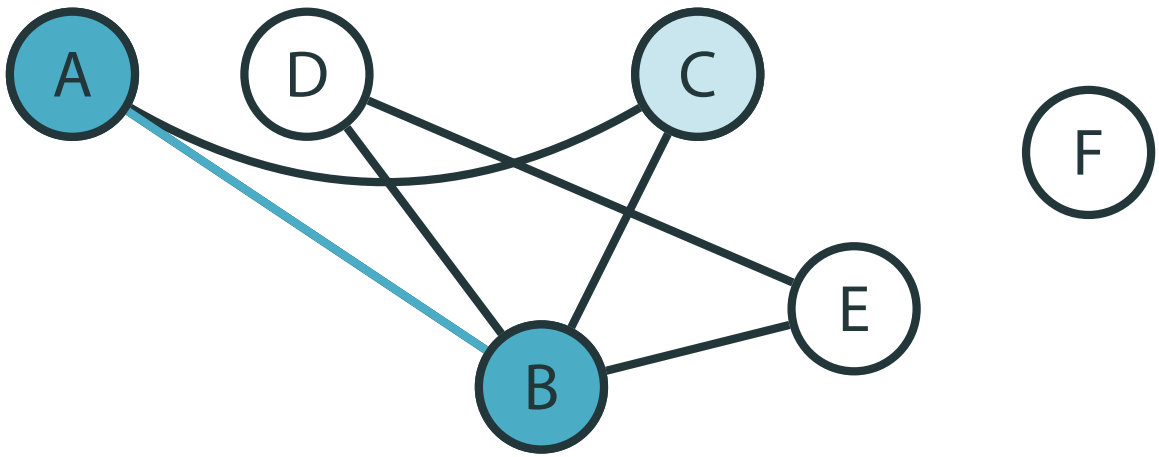
```
def Explore(v):
    visited[v]=True

    for u in graph[v]:
        if not visited[u]:
            Explore(u)
```



Поиск в глубину

- Это можно описать рекурсивно

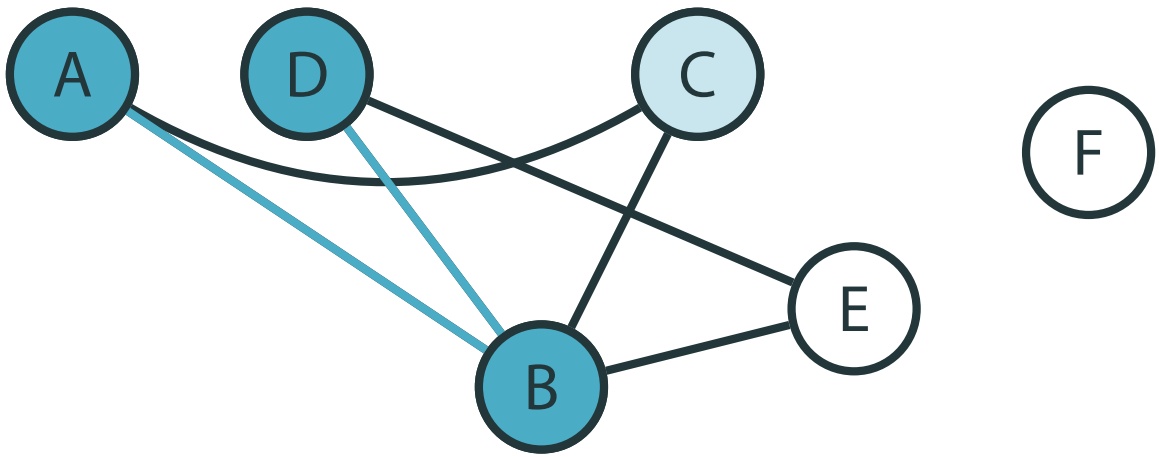
```
def Explore(v):
    visited[v]=True

    for u in graph[v]:
        if not visited[u]:
            Explore(u)
```



Поиск в глубину

- Это можно описать рекурсивно

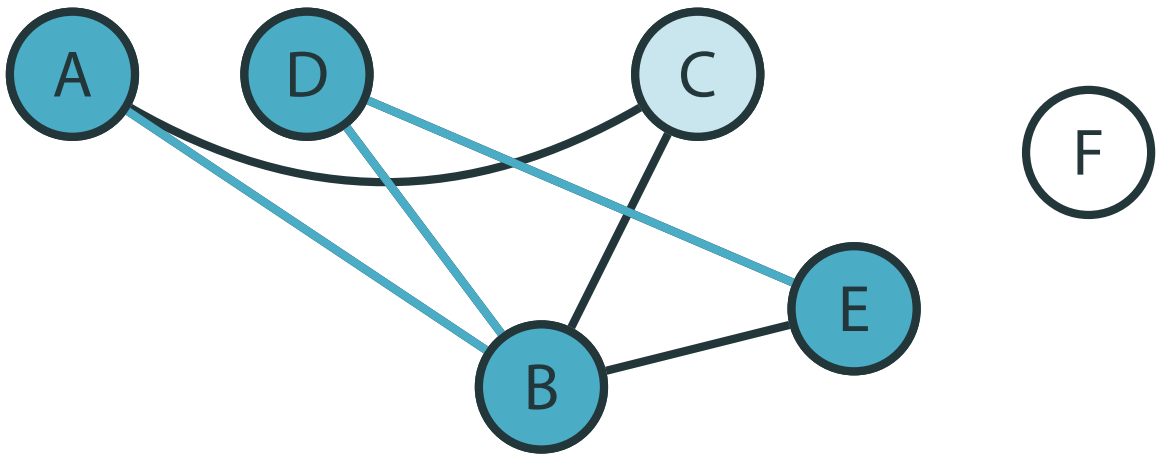
```
def Explore(v):
    visited[v]=True

    for u in graph[v]:
        if not visited[u]:
            Explore(u)
```



Поиск в глубину

- Это можно описать рекурсивно

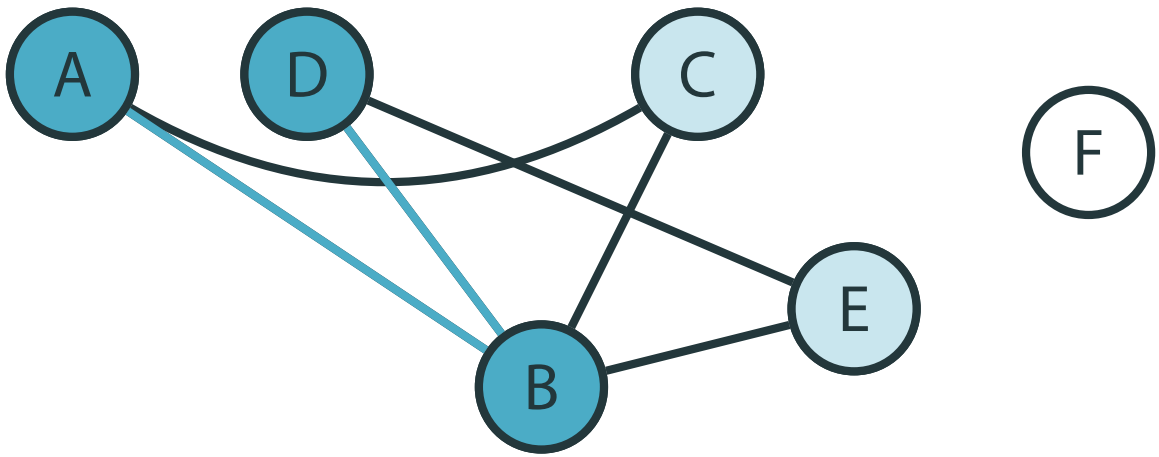
```
def Explore(v):
    visited[v]=True

    for u in graph[v]:
        if not visited[u]:
            Explore(u)
```



Поиск в глубину

- Это можно описать рекурсивно

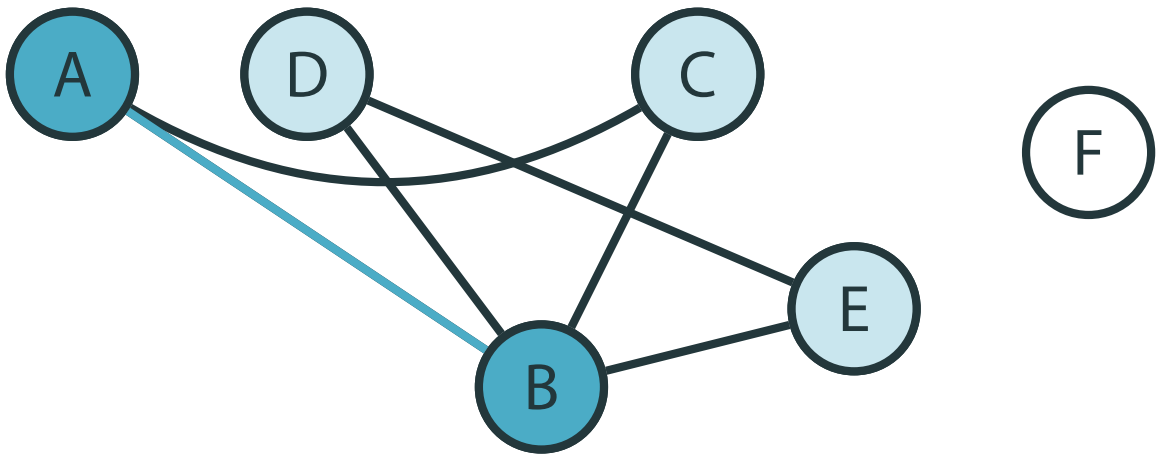
```
def Explore(v):
    visited[v]=True

    for u in graph[v]:
        if not visited[u]:
            Explore(u)
```



Поиск в глубину

- Это можно описать рекурсивно

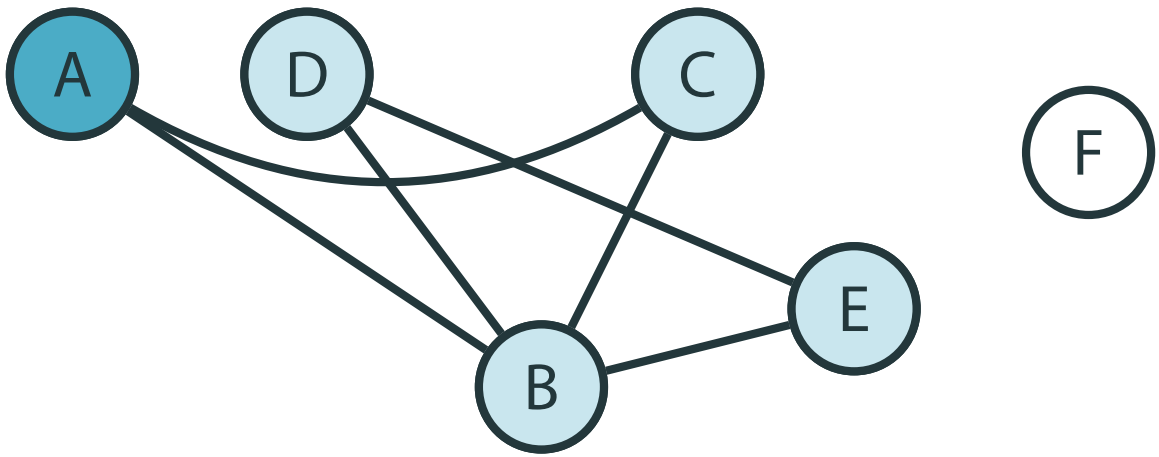
```
def Explore(v):
    visited[v]=True

    for u in graph[v]:
        if not visited[u]:
            Explore(u)
```



Поиск в глубину

- Это можно описать рекурсивно

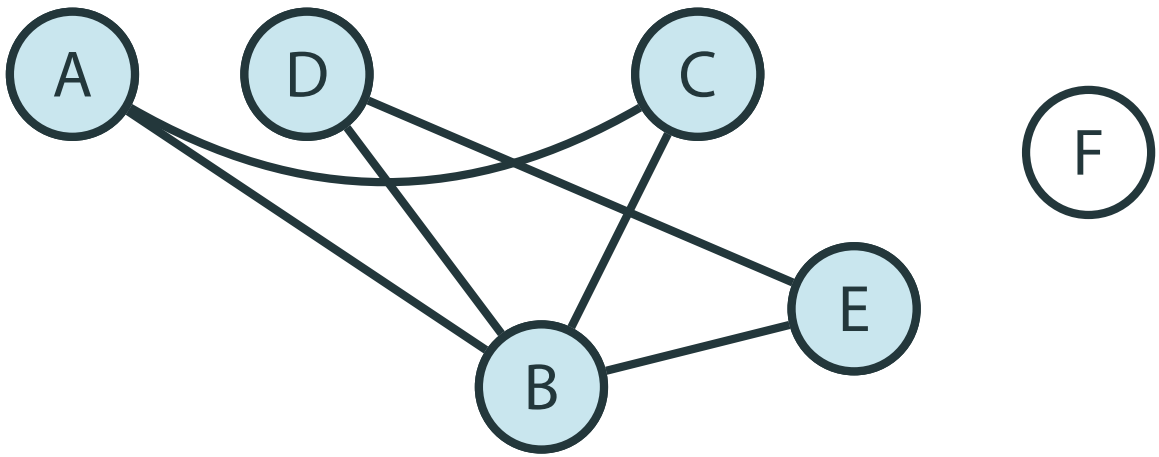
```
def Explore(v):
    visited[v]=True

    for u in graph[v]:
        if not visited[u]:
            Explore(u)
```



Поиск в глубину

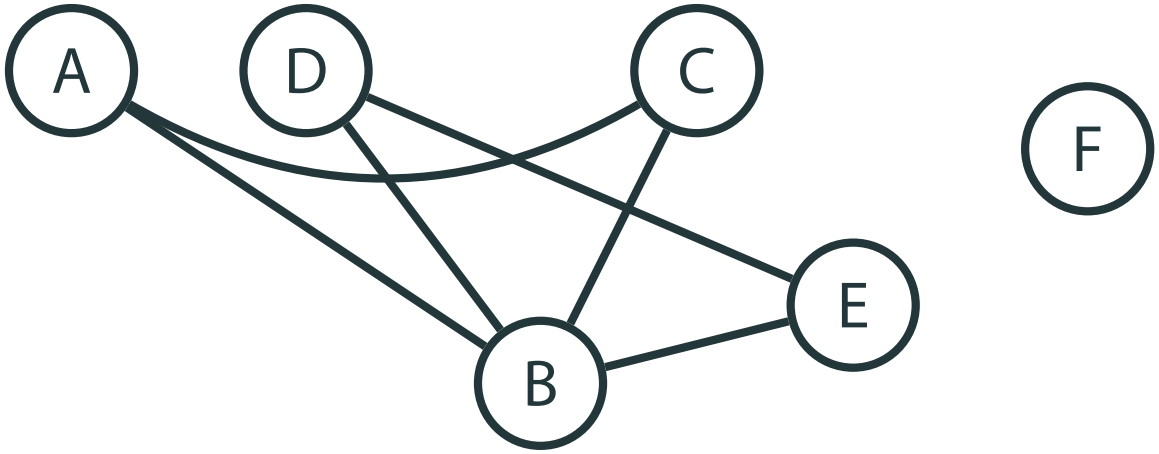
- Это можно описать рекурсивно

```
def Explore(v):
    visited[v]=True

    for u in graph[v]:
        if not visited[u]:
            Explore(u)
```



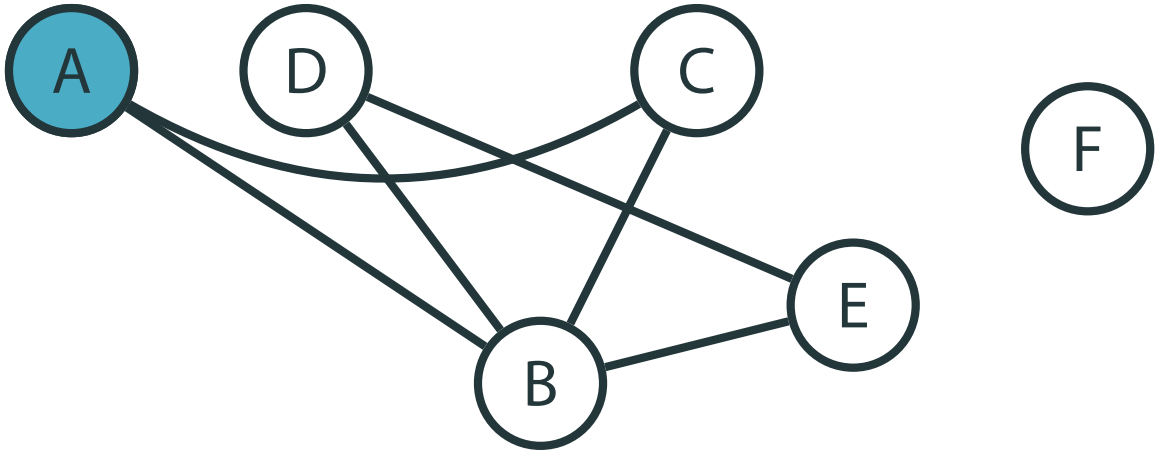
Поиск в глубину

По существу, у нас поддерживается стек рассматриваемых в данный момент вершин



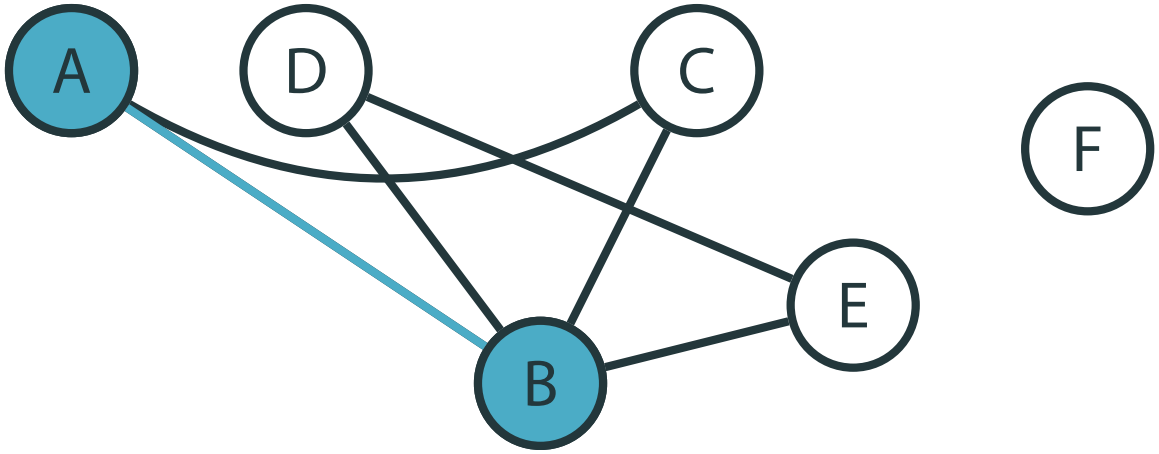
Поиск в глубину

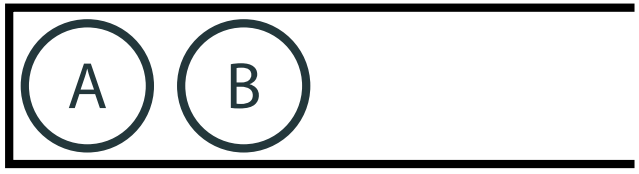
По существу, у нас поддерживается стек рассматриваемых в данный момент вершин



Поиск в глубину

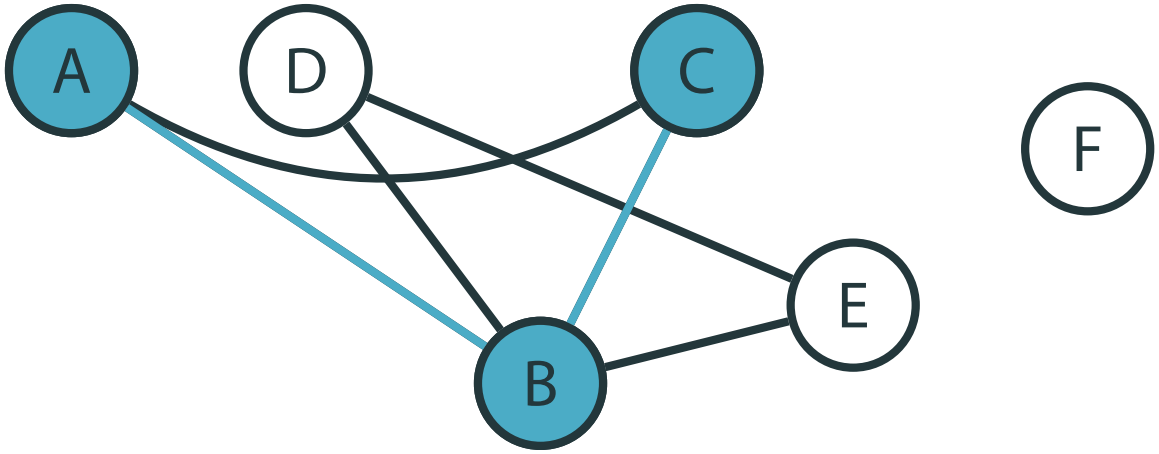
По существу, у нас поддерживается стек рассматриваемых в данный момент вершин

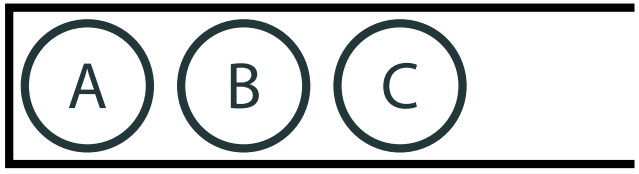




Поиск в глубину

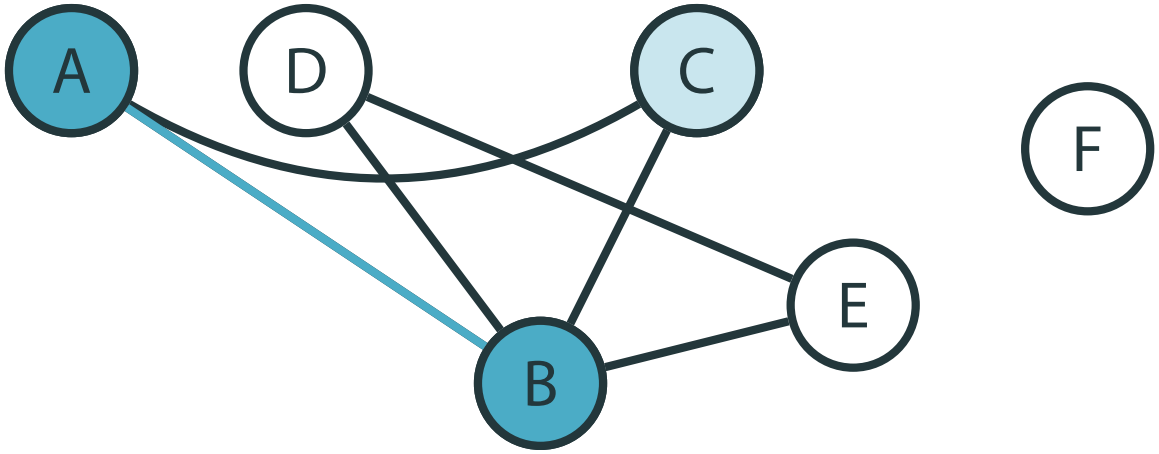
По существу, у нас поддерживается стек рассматриваемых в данный момент вершин

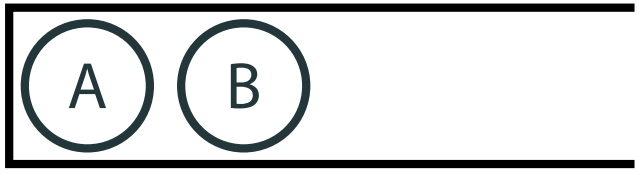




Поиск в глубину

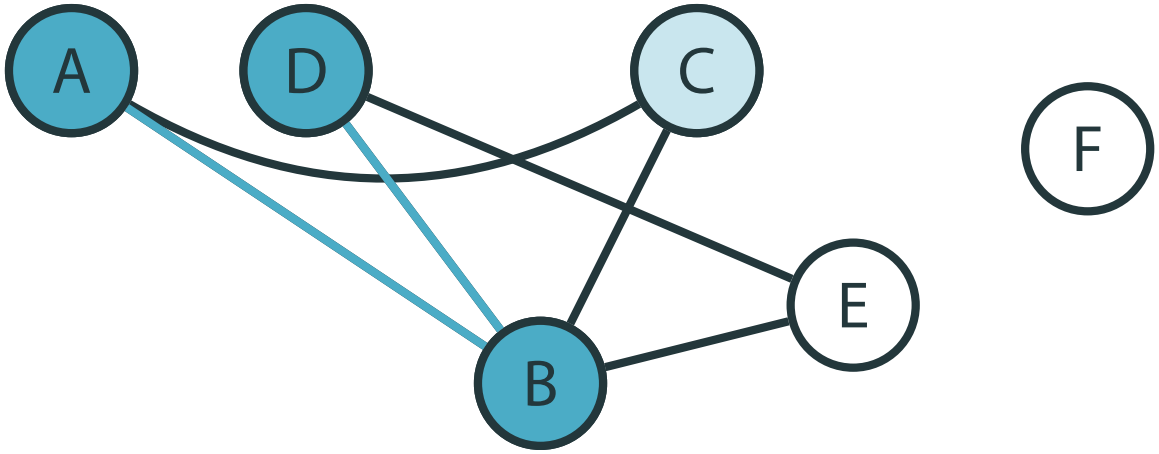
По существу, у нас поддерживается стек рассматриваемых в данный момент вершин

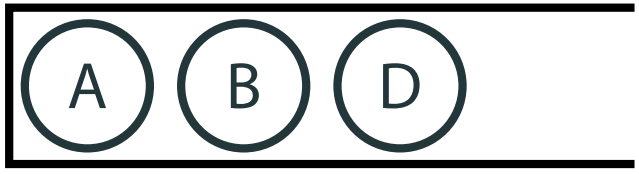




Поиск в глубину

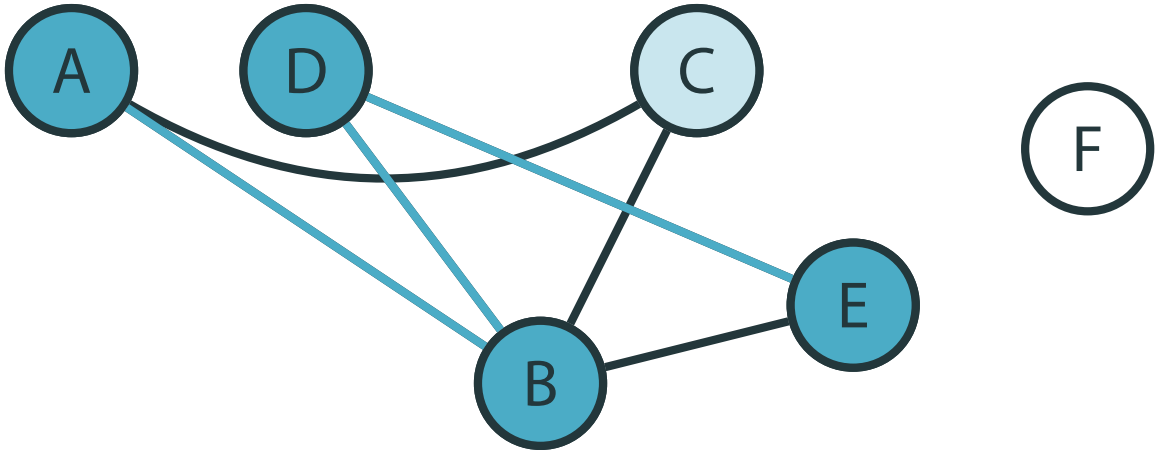
По существу, у нас поддерживается стек рассматриваемых в данный момент вершин

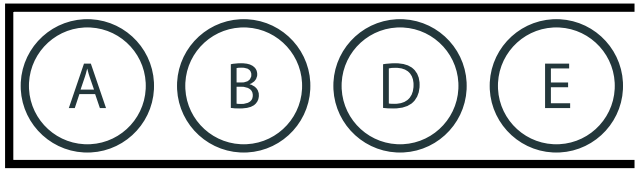




Поиск в глубину

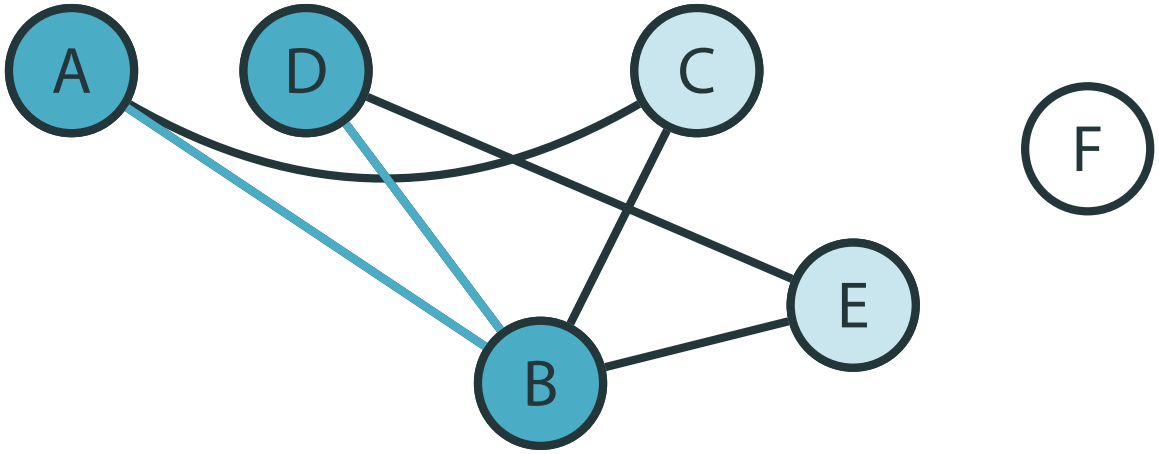
По существу, у нас поддерживается стек рассматриваемых в данный момент вершин

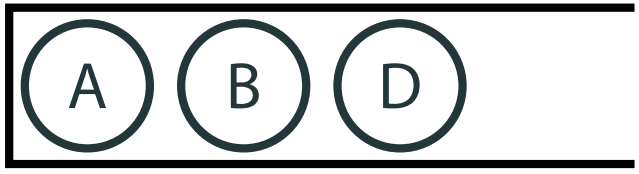




Поиск в глубину

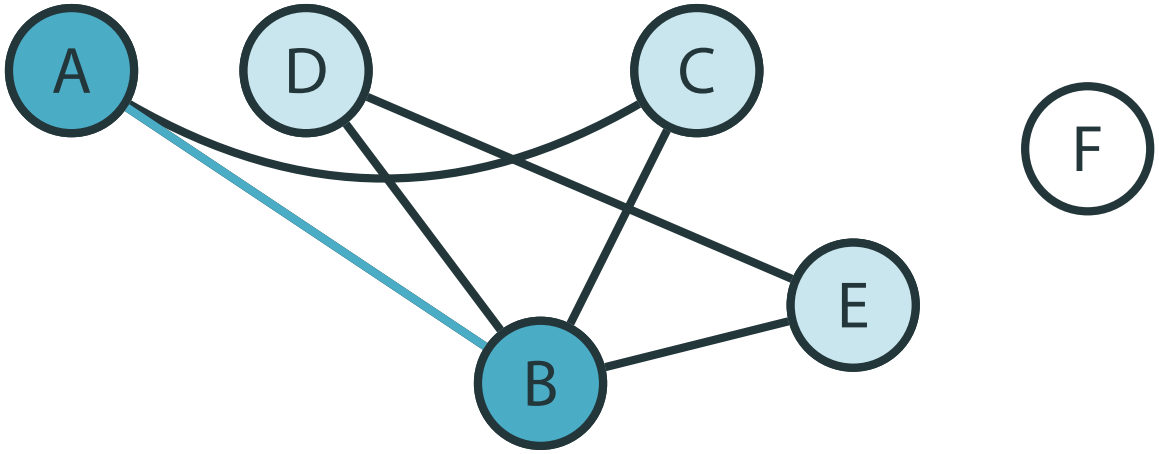
По существу, у нас поддерживается стек рассматриваемых в данный момент вершин

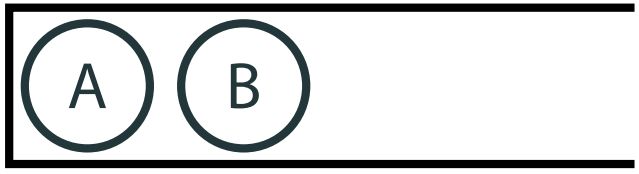




Поиск в глубину

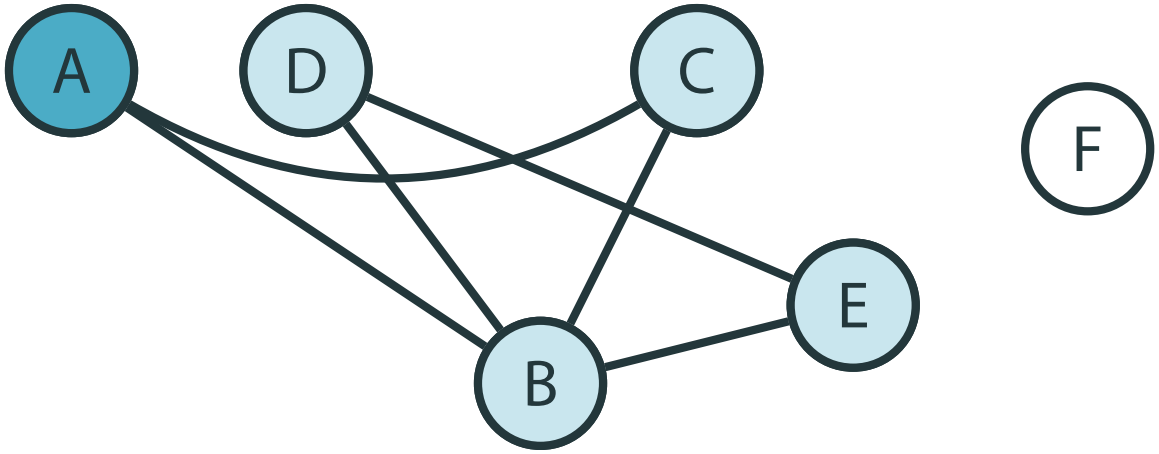
По существу, у нас поддерживается стек рассматриваемых в данный момент вершин





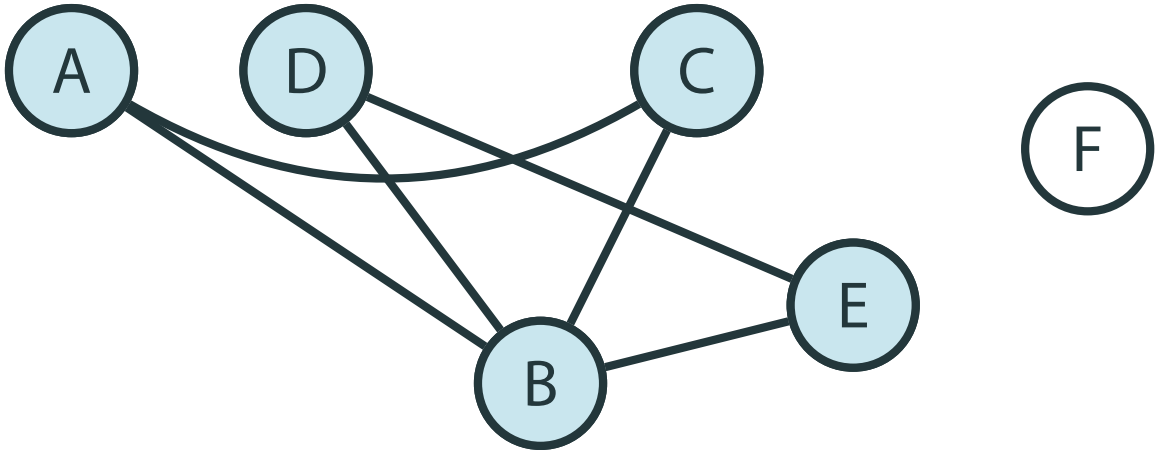
Поиск в глубину

По существу, у нас поддерживается стек рассматриваемых в данный момент вершин



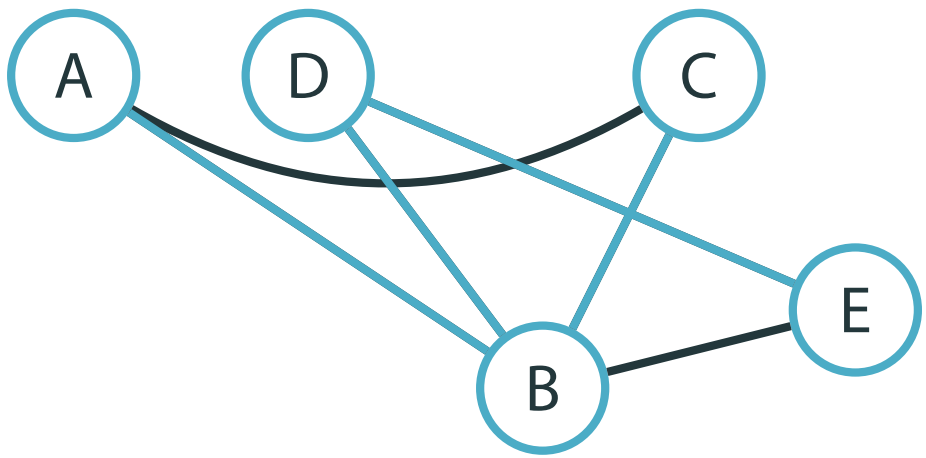
Поиск в глубину

По существу, у нас поддерживается стек рассматриваемых в данный момент вершин



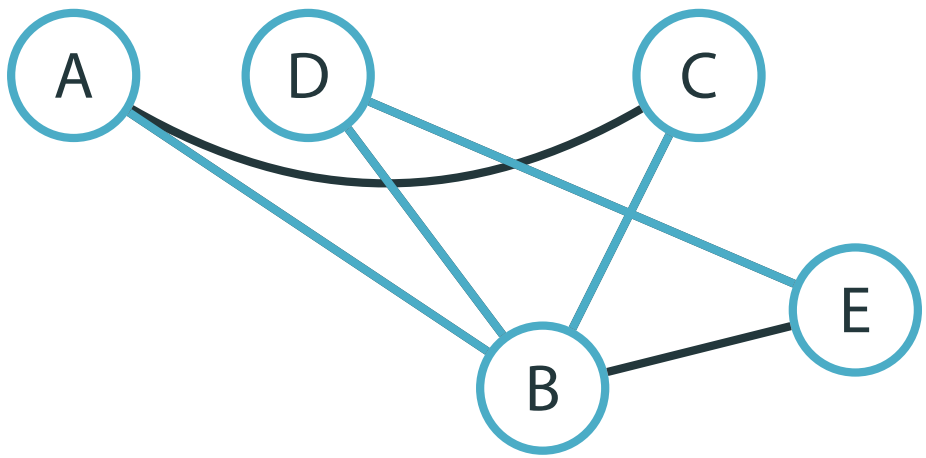
Поиск в глубину

- Поиск в глубину обходит по дереву



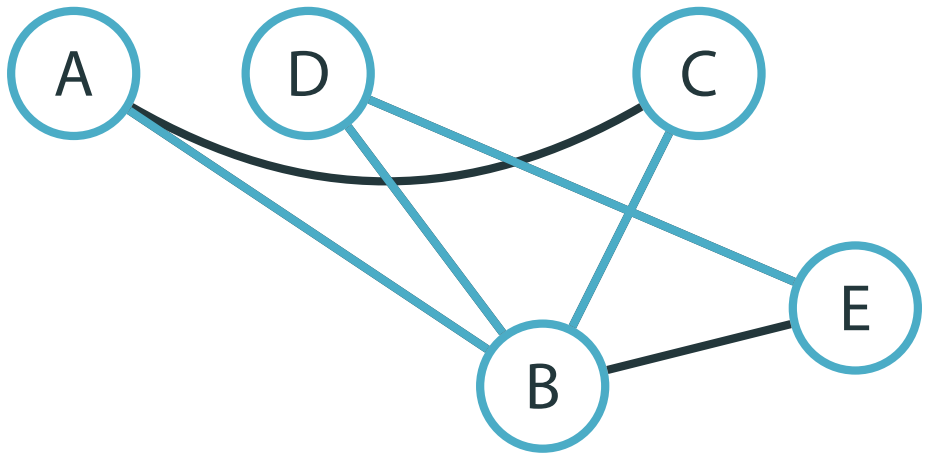
Поиск в глубину

- Поиск в глубину обходит по дереву
- Еще раз доказали, что в связном графе есть остовное дерево



Поиск в глубину

- Поиск в глубину обходит по дереву
- Еще раз доказали, что в связном графе есть остовное дерево
- Так можно эффективно находить остовное дерево



Поиск в глубину

- Что делает процедура Explore?

Поиск в глубину

- Что делает процедура Explore?
- Обходит все вершины компоненты связности, содержащей v

Поиск в глубину

- Что делает процедура Explore?
- Обходит все вершины компоненты связности, содержащей v
- Что делать, если хотим обойти все вершины графа?

Поиск в глубину

- Запускать Explore заново для еще не посещенных вершин

```
def dfs():
    for v in graph:
        if not visited[v]:
            Explore(v)
```

Поиск в глубину

- Запускать Explore заново для еще не посещенных вершин
- Эта процедура называется поиском в глубину

```
def dfs():
    for v in graph:
        if not visited[v]:
            Explore(v)
```

Поиск в глубину

- Как быстро работает поиск в глубину?

Поиск в глубину

- Как быстро работает поиск в глубину?
- Для каждой вершины мы

Поиск в глубину

- Как быстро работает поиск в глубину?
- Для каждой вершины мы
    1. обрабатываем ее

Поиск в глубину

- Как быстро работает поиск в глубину?
- Для каждой вершины мы
    1. обрабатываем ее
    2. перебираем соседей

Поиск в глубину

- Как быстро работает поиск в глубину?
- Для каждой вершины мы
    1. обрабатываем ее
    2. перебираем соседей
- Первое — константа операций на одну вершину

Поиск в глубину

- Как быстро работает поиск в глубину?
- Для каждой вершины мы
    1. обрабатываем ее
    2. перебираем соседей
- Первое — константа операций на одну вершину
- Второе — каждое ребро просматриваем два раза

Поиск в глубину

- Как быстро работает поиск в глубину?
- Для каждой вершины мы
    1. обрабатываем ее
    2. перебираем соседей
- Первое — константа операций на одну вершину
- Второе — каждое ребро просматриваем два раза
- Это константа операций для каждого ребра

Поиск в глубину

- Как быстро работает поиск в глубину?
- Для каждой вершины мы
    1. обрабатываем ее
    2. перебираем соседей
- Первое — константа операций на одну вершину
- Второе — каждое ребро просматриваем два раза
- Это константа операций для каждого ребра
- Число операций порядка $|V| + |E|$, умноженного на константу

Поиск в глубину

- Как использовать поиск в глубину?

Поиск в глубину

- Как использовать поиск в глубину?
- Обработка вершины до и после запуска рекурсии в ней

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

Поиск в глубину

- Например, можно фиксировать время, до начала обработки вершины и после

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

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

Поиск в глубину

- Например, можно фиксировать время, до начала обработки вершины и после
- Для любой вершины v у нас будет 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

- Очень полезно для анализа графов

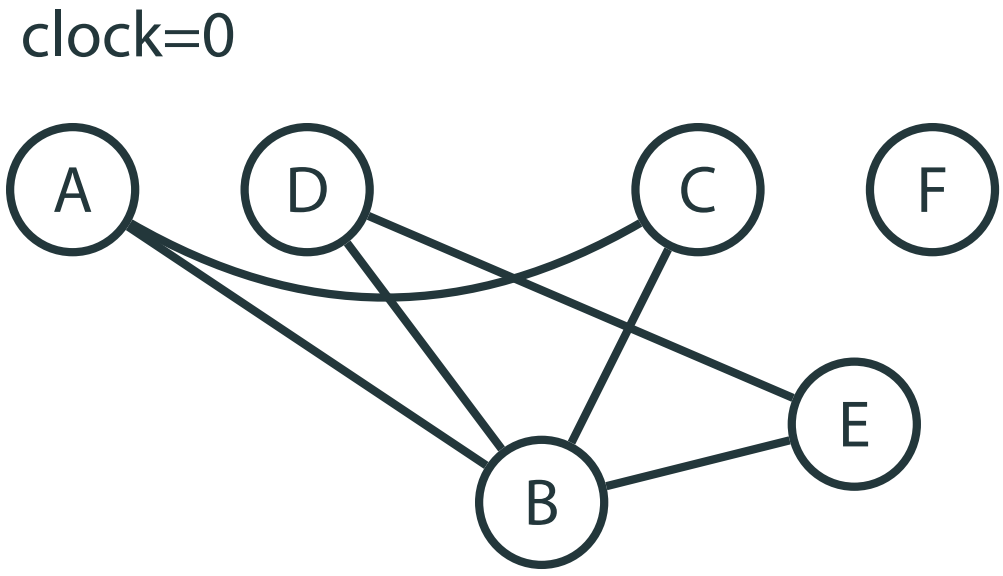
```
def Previsit(v):
    pre[v]=clock
    clock+=1

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

Поиск в глубину

- Очень полезно для анализа графов
- Увидим примеры позже

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

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

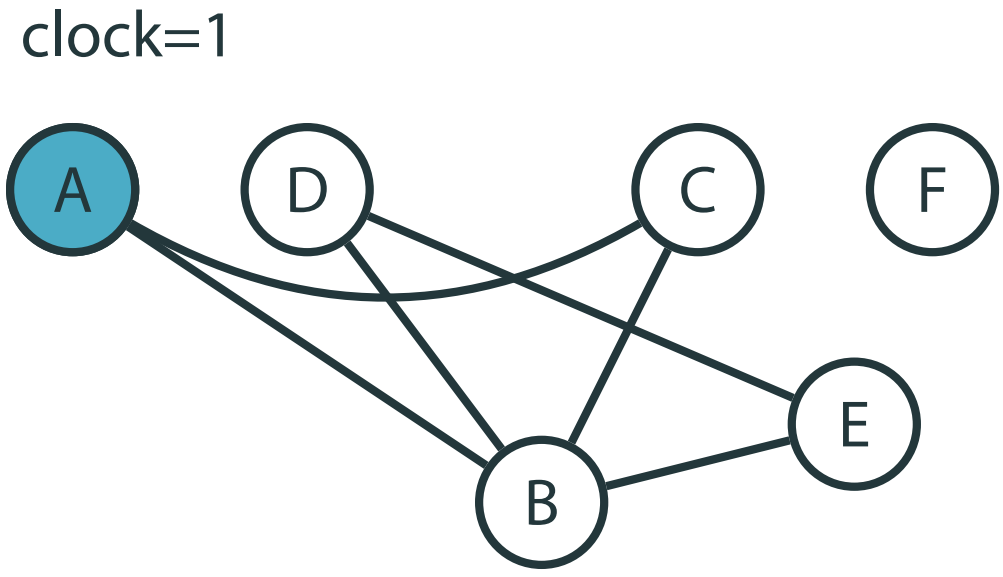
Поиск в глубину




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

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

	pre	post
A		
B		
C		
D		
E		
F		

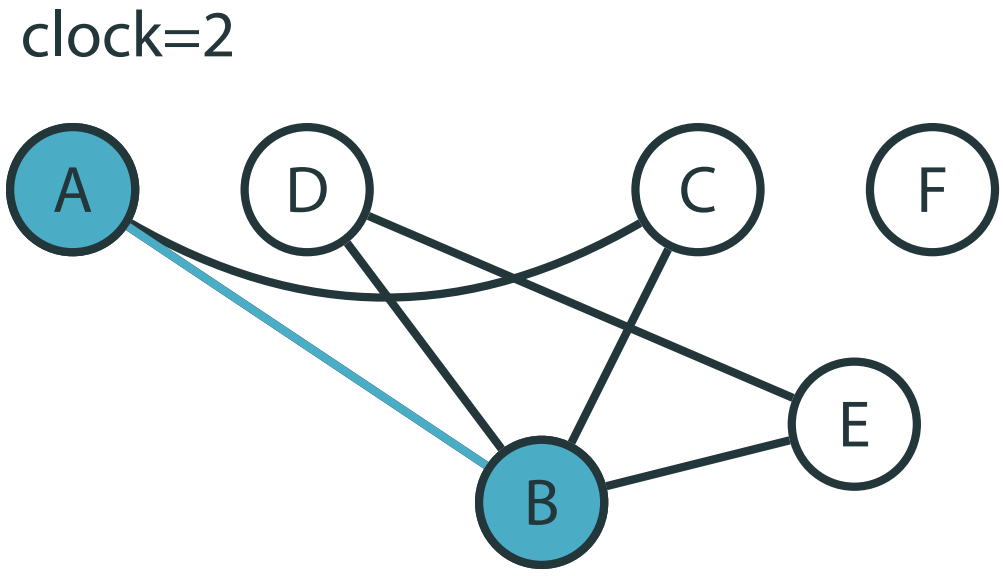
Поиск в глубину




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

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

	pre	post
A	0	
B		
C		
D		
E		
F		

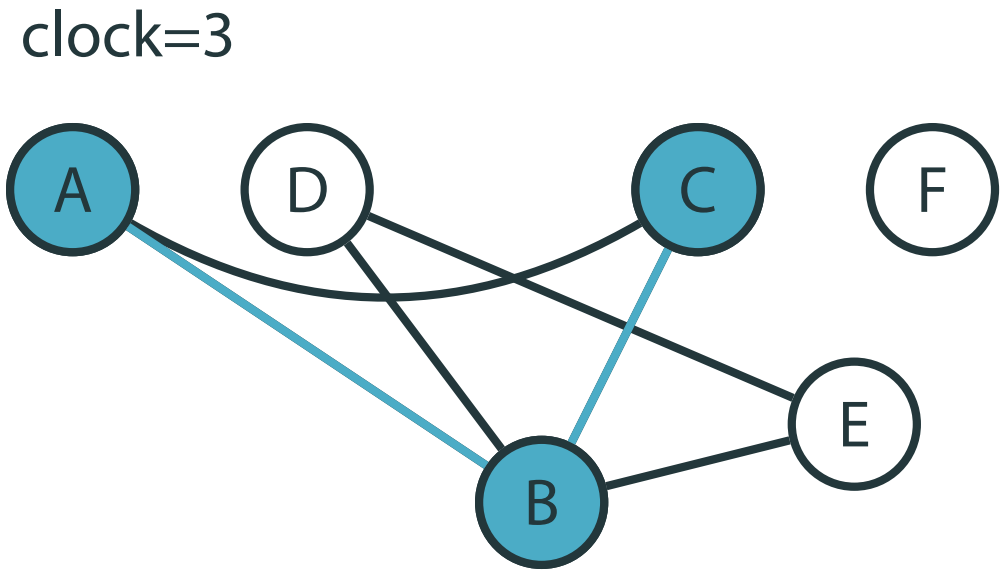
Поиск в глубину




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

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

	pre	post
A	0	
B	1	
C		
D		
E		
F		

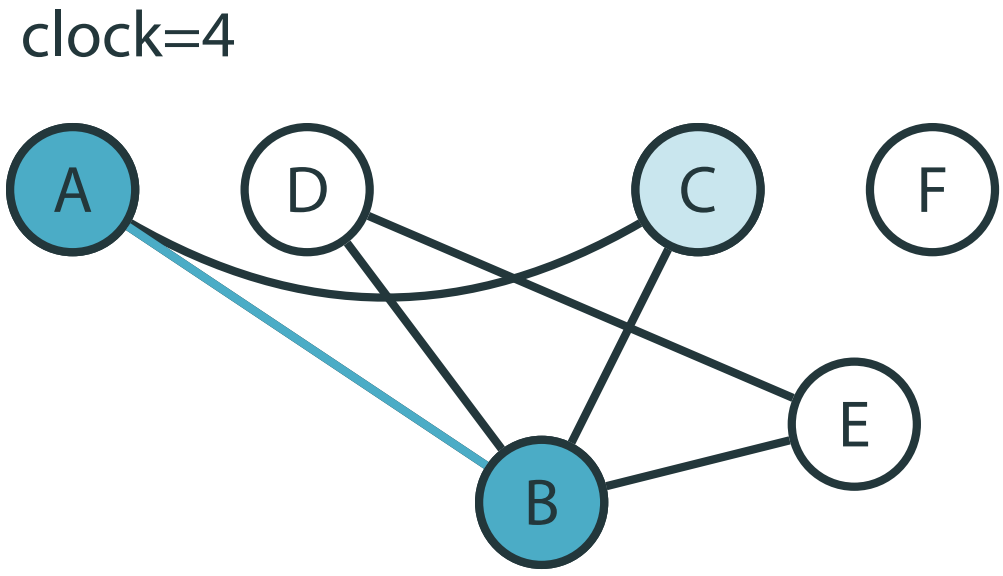
Поиск в глубину




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

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

	pre	post
A	0	
B	1	
C	2	
D		
E		
F		

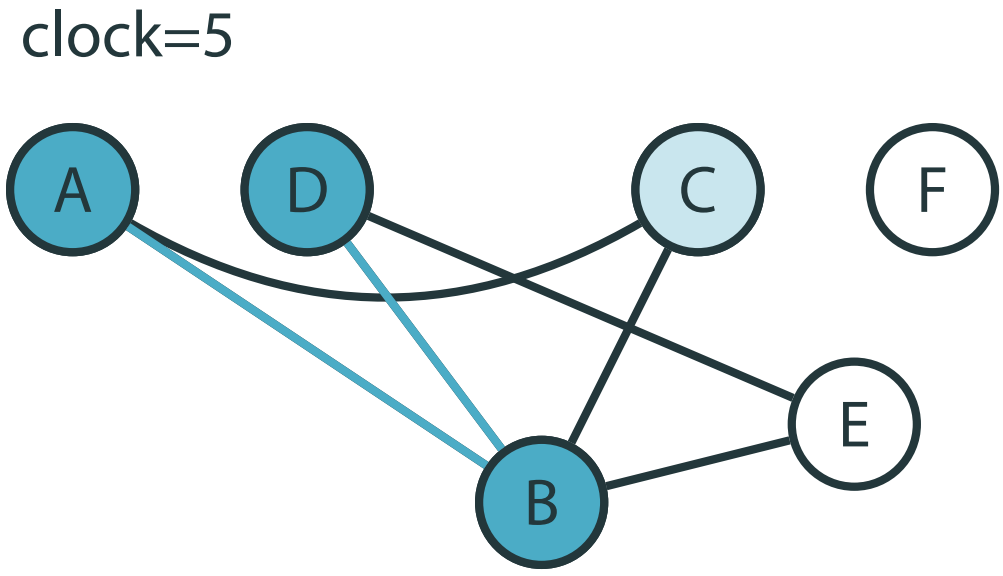
Поиск в глубину




```
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		
E		
F		

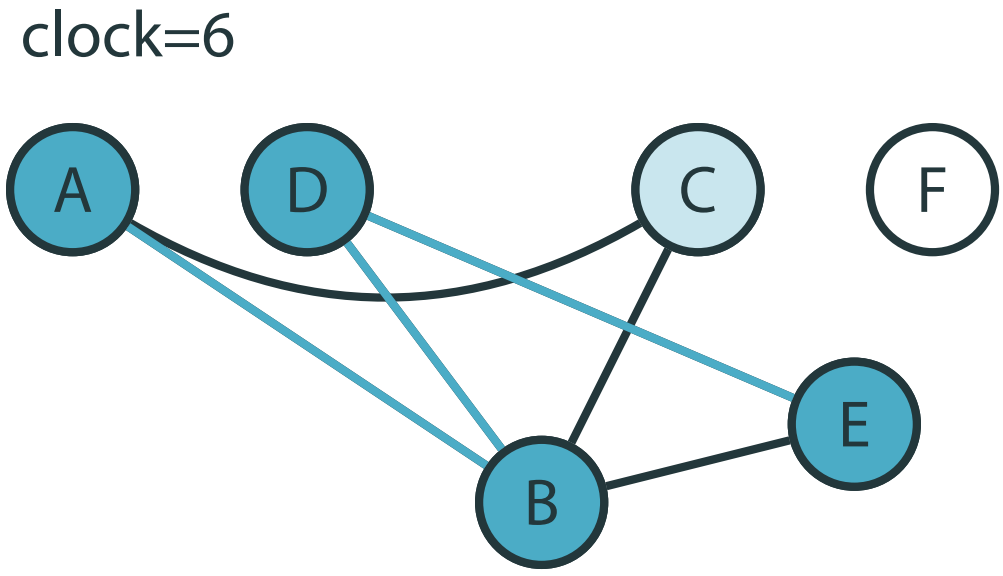
Поиск в глубину




```
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		

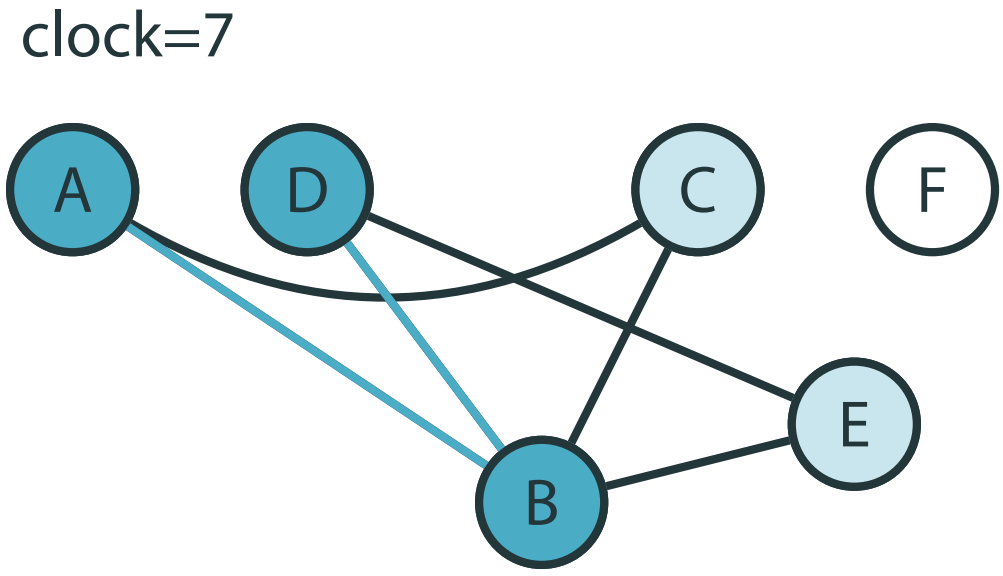
Поиск в глубину




```
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	
F		

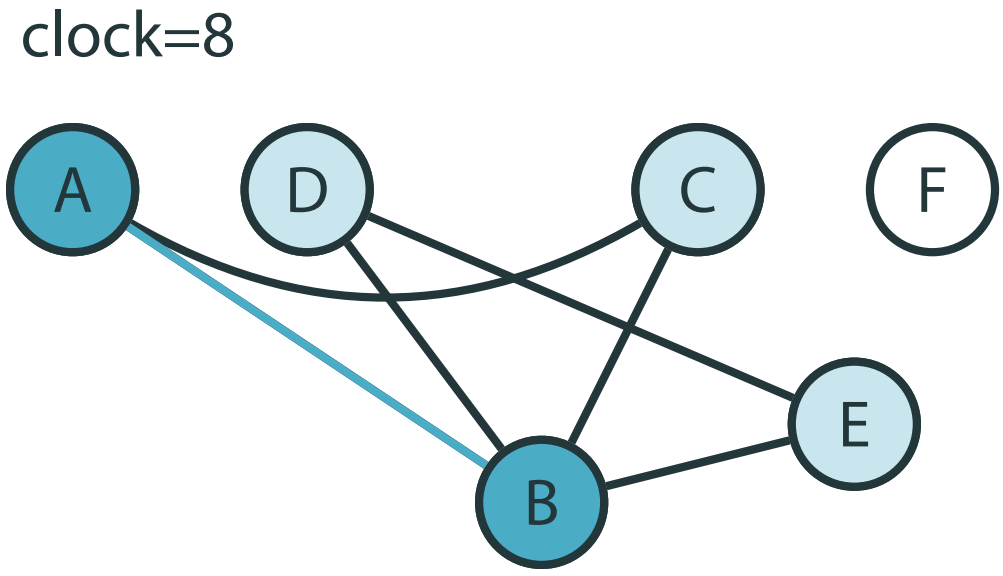
Поиск в глубину




```
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	5	6
F		

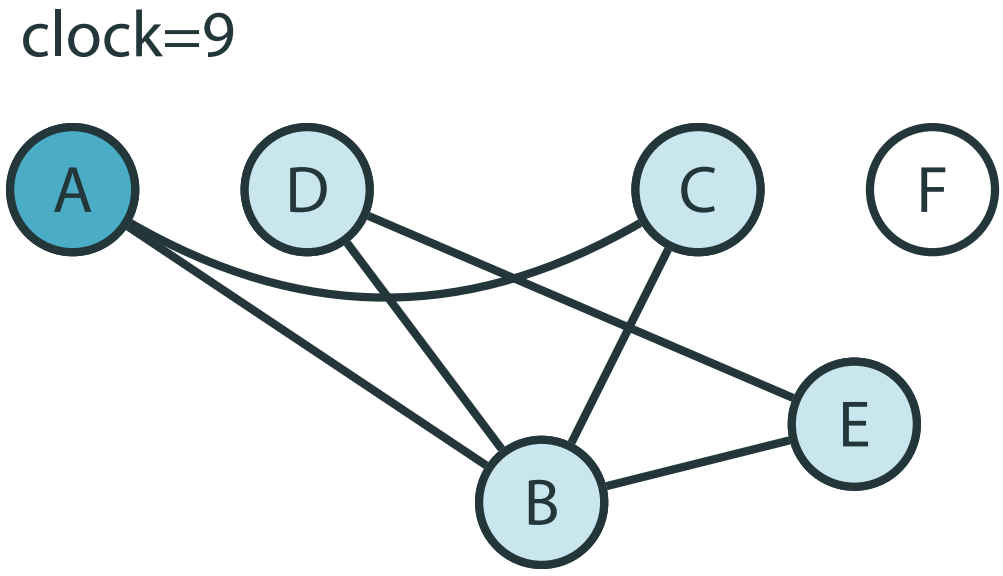
Поиск в глубину




```
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		

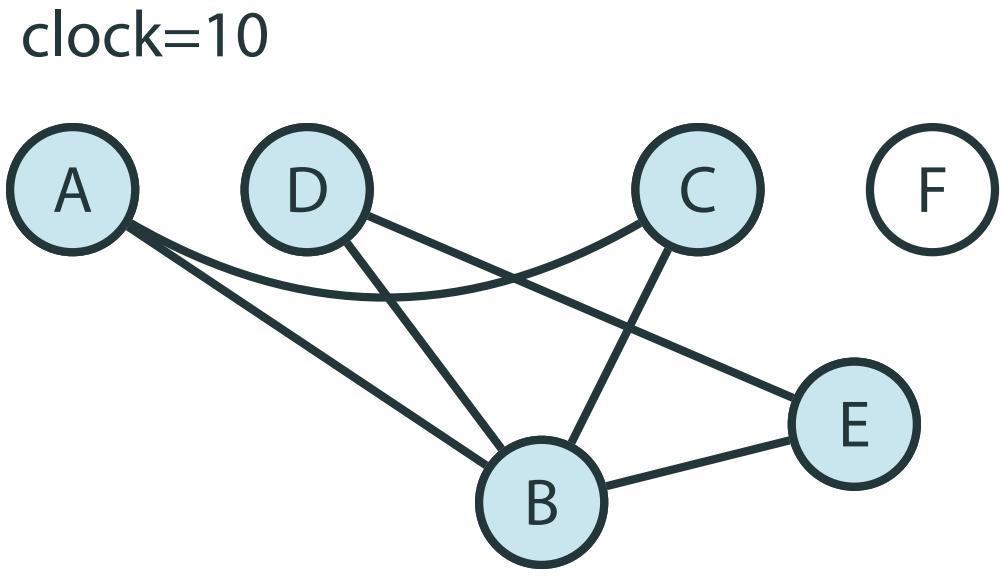
Поиск в глубину




```
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		

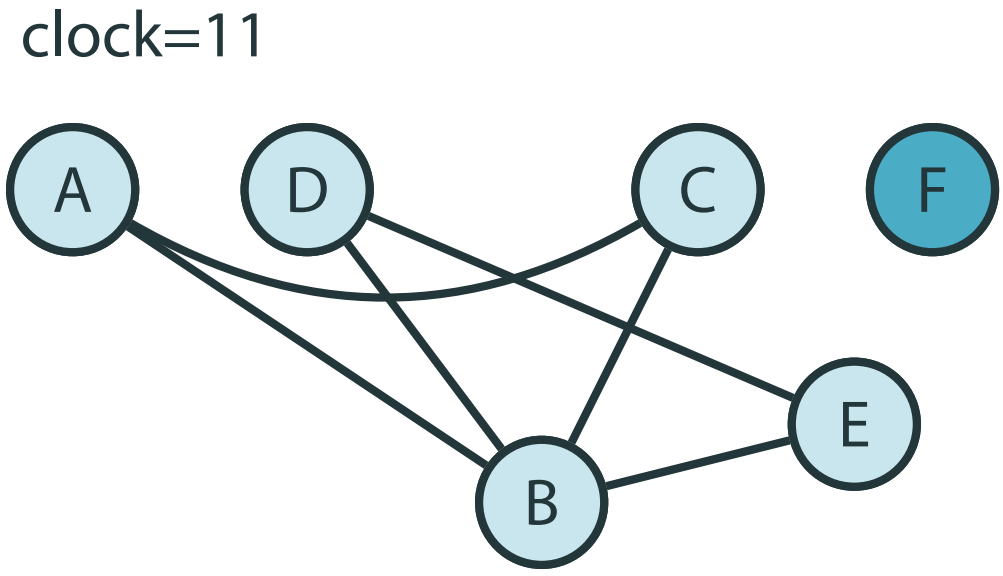
Поиск в глубину




```
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		

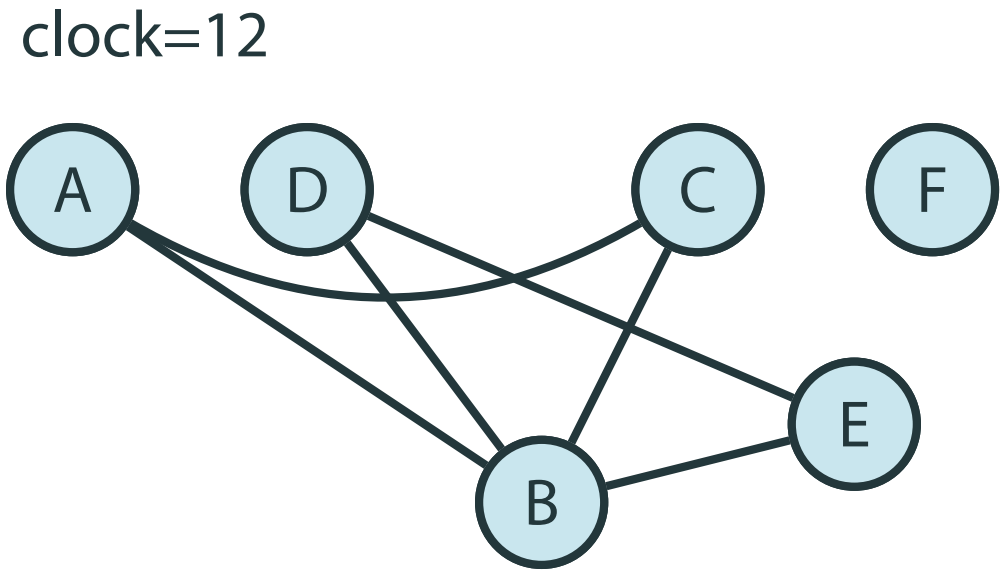
Поиск в глубину




```
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

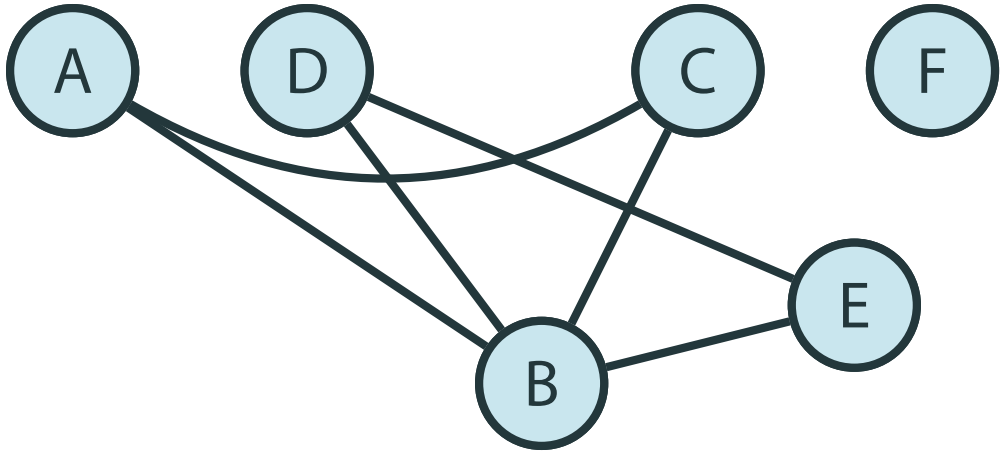



```
def Previsit(v):
    pre[v]=clock
    clock+=1

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

clock=12



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

Отрезки $[pre[v], post[v]]$ либо вложены, либо не пересекаются

Поиск в глубину

- Итак, для каждое вершины v мы нашли отрезок $[pre[v], post[v]]$

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

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

Поиск в глубину

- Итак, для каждое вершины v мы нашли отрезок $[pre[v], post[v]]$
- Эти отрезки либо вложены, либо не пересекаются

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

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

Поиск в глубину

- Итак, для каждое вершины v мы нашли отрезок $[pre[v], post[v]]$
- Эти отрезки либо вложены, либо не пересекаются
- Они пригодятся нам при рассмотрении ориентированных графов

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

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