# Пути и достижимость

Владимир Подольский

Факультет компьютерных наук, Высшая Школа Экономики

# Пути и достижимость

Пути и достижимость

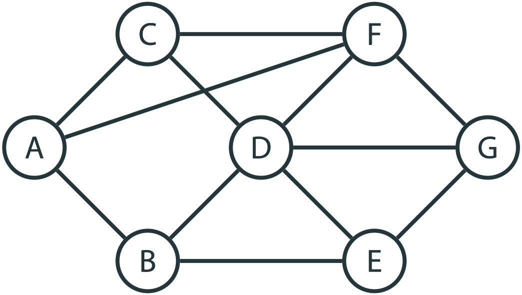
Число компонент связности

Обходы и поиск в графах

Расстояния в графах

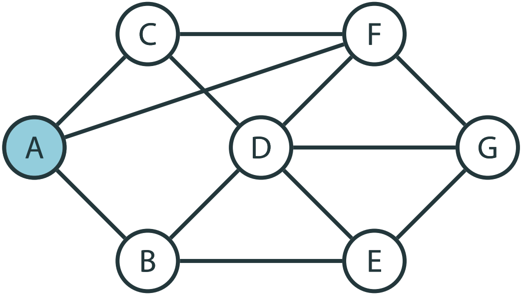
# Пути в графах

- Бывает полезно рассматривать пути в графах



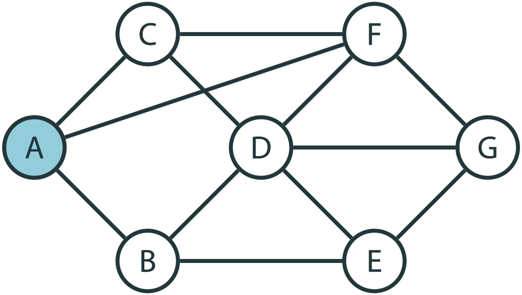
# Пути в графах

- Бывает полезно рассматривать пути в графах
- Начинаем с какой-то вершины



# Пути в графах

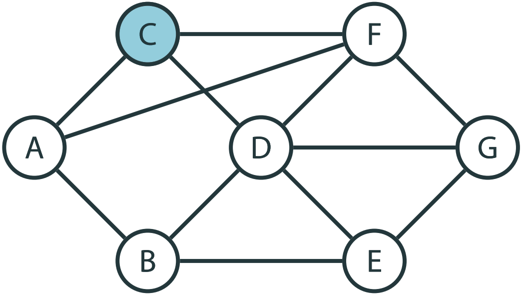
- Бывает полезно рассматривать пути в графах
- Начинаем с какой-то вершины
- На каждом шаге можем перейти по ребру в следующую вершину




Путь: A

# Пути в графах

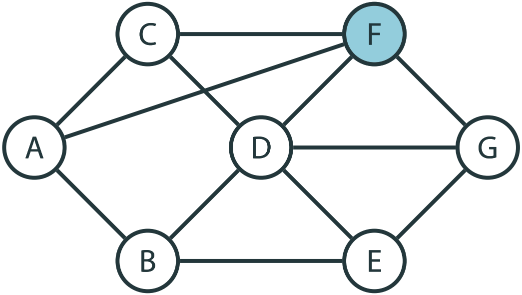
- Бывает полезно рассматривать пути в графах
- Начинаем с какой-то вершины
- На каждом шаге можем перейти по ребру в следующую вершину




Путь: A, C

# Пути в графах

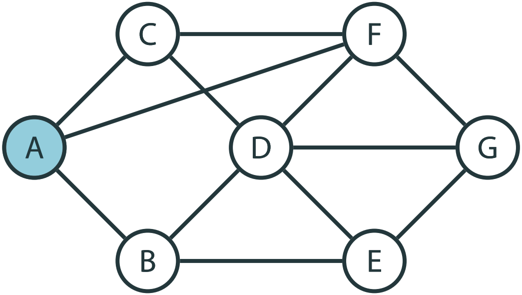
- Бывает полезно рассматривать пути в графах
- Начинаем с какой-то вершины
- На каждом шаге можем перейти по ребру в следующую вершину




Путь: A, C, F

# Пути в графах

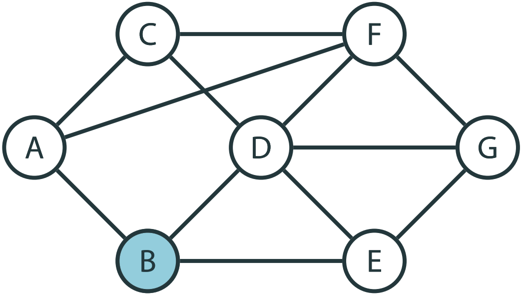
- Бывает полезно рассматривать пути в графах
- Начинаем с какой-то вершины
- На каждом шаге можем перейти по ребру в следующую вершину




Путь: A, C, F, A

# Пути в графах

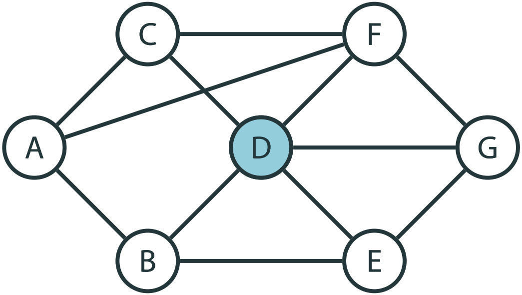
- Бывает полезно рассматривать пути в графах
- Начинаем с какой-то вершины
- На каждом шаге можем перейти по ребру в следующую вершину




Путь: A, C, F, A, B

# Пути в графах

- Бывает полезно рассматривать пути в графах
- Начинаем с какой-то вершины
- На каждом шаге можем перейти по ребру в следующую вершину




Путь: A, C, F, A, B, D

# Пути в графах

- Бывает, что пути естественно возникают в изначальной задаче

# Пути в графах

- Бывает, что пути естественно возникают в изначальной задаче
- Например, в транспортных графах

# Пути в графах

- Бывает, что пути естественно возникают в изначальной задаче
- Например, в транспортных графах
- Бывает, что пути полезны для анализа графа

# Пути в графах

- Бывает, что пути естественно возникают в изначальной задаче
- Например, в транспортных графах
- Бывает, что пути полезны для анализа графа
- Например, в графах социальных сетей для анализа окружения пользователя

# Пути в графах

- Формально путь это последовательность вершин: $v_0, v_1, \dots, v_k$

# Пути в графах

- Формально путь это последовательность вершин: $v_0, v_1, \ldots, v_k$
- Из каждой вершины есть ребро в следующую

# Пути в графах

- Формально путь это последовательность вершин: $v_0, v_1, \dots, v_k$
- Из каждой вершины есть ребро в следующую
- Длина пути — число шагов в нем

# Пути в графах

- Формально путь это последовательность вершин: $v_0, v_1, \ldots, v_k$
- Из каждой вершины есть ребро в следующую
- Длина пути — число шагов в нем
- В наших обозначениях длина пути $k$

# Пути в графах

- Формально путь это последовательность вершин: $v_0, v_1, \ldots, v_k$
- Из каждой вершины есть ребро в следующую
- Длина пути — число шагов в нем
- В наших обозначениях длина пути $k$
- Вершины могут повторяться

# Пути в графах

- Формально путь это последовательность вершин: $v_0, v_1, \dots, v_k$
- Из каждой вершины есть ребро в следующую
- Длина пути — число шагов в нем
- В наших обозначениях длина пути $k$
- Вершины могут повторяться
- Если вершины не повторяются, то это простой путь

# Циклы в графах

- Если начальная вершина пути совпадает с конечной, то это цикл: $v_0, v_1, \dots, v_k = v_0$

# Циклы в графах

- Если начальная вершина пути совпадает с конечной, то это цикл: $v_0, v_1, \dots, v_k = v_0$
- Длина цикла — число шагов в нем (у нас $k$)

# Циклы в графах

- Если начальная вершина пути совпадает с конечной, то это цикл: $v_0, v_1, \ldots, v_k = v_0$
- Длина цикла — число шагов в нем (у нас $k$)
- Простой цикл — нет повторов вершин, длина не меньше 3

# Циклы в графах

- Если начальная вершина пути совпадает с конечной, то это цикл: $v_0, v_1, \ldots, v_k = v_0$
- Длина цикла — число шагов в нем (у нас $k$)
- Простой цикл — нет повторов вершин, длина не меньше 3
- Естественно возникают в транспортных графах

# Циклы в графах

- Если начальная вершина пути совпадает с конечной, то это цикл: $v_0, v_1, \ldots, v_k = v_0$
- Длина цикла — число шагов в нем (у нас $k$)
- Простой цикл — нет повторов вершин, длина не меньше 3
- Естественно возникают в транспортных графах
- Важны при обходах графов

# Циклы в графах

- Если начальная вершина пути совпадает с конечной, то это цикл: $v_0, v_1, \ldots, v_k = v_0$
- Длина цикла — число шагов в нем (у нас $k$)
- Простой цикл — нет повторов вершин, длина не меньше 3
- Естественно возникают в транспортных графах
- Важны при обходах графов
- Про циклы полезно помнить при работе с графами

# Циклы в графах

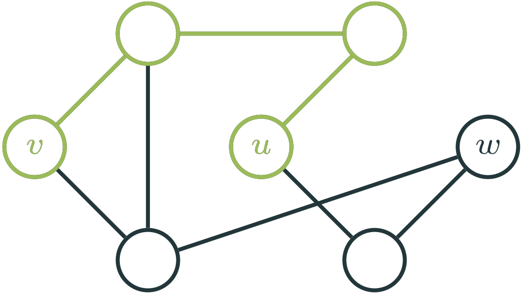
- Если начальная вершина пути совпадает с конечной, то это цикл: $v_0, v_1, \ldots, v_k = v_0$
- Длина цикла — число шагов в нем (у нас $k$)
- Простой цикл — нет повторов вершин, длина не меньше 3
- Естественно возникают в транспортных графах
- Важны при обходах графов
- Про циклы полезно помнить при работе с графами
- Они могут создавать проблемы для алгоритмов на графах

# Заглушка

Это слайд-заглушка, нужен только чтобы не съехала нумерация

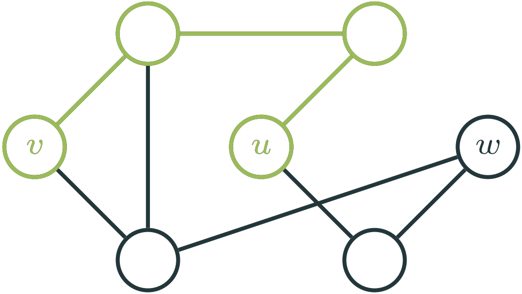
# Достижимость

- Вершина $u$ достижима из вершины $v$, если есть путь из $v$ в $u$



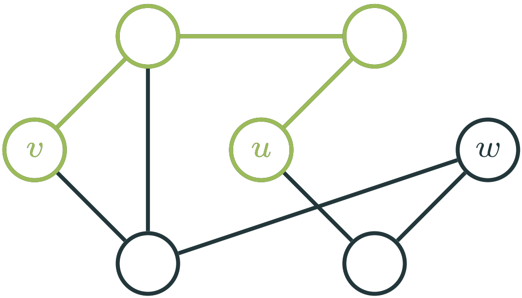
# Достижимость

- Вершина $u$ достижима из вершины $v$, если есть путь из $v$ в $u$
- Это симметрично, если $u$ достижима из $v$, то и $v$ достижима из $u$



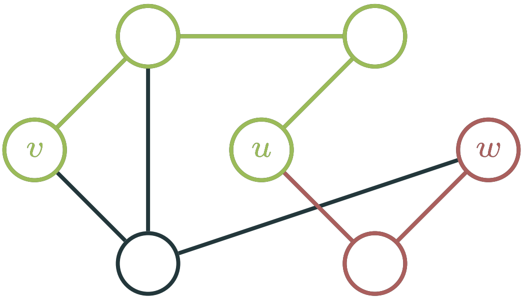
# Достижимость

- Также говорим, что вершины $u$ и $v$ связаны



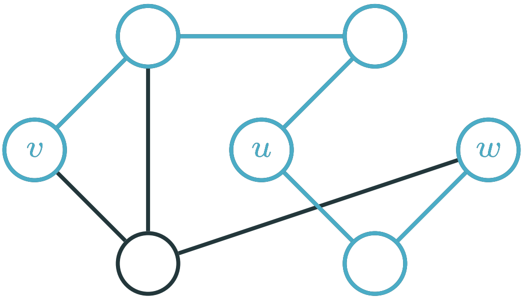
# Достижимость

- Также говорим, что вершины $u$ и $v$ связаны
- Это транзитивно: если $u$ достижима из $v$, а $w$ достижима из $u$, то $w$ достижима из $u$



# Достижимость

- Также говорим, что вершины $u$ и $v$ связаны
- Это транзитивно: если $u$ достижима из $v$, а $w$ достижима из $u$, то $w$ достижима из $u$



# Связность

- Важное свойство графа: можно ли из всякой вершины дойти в любую другую?

# Связность

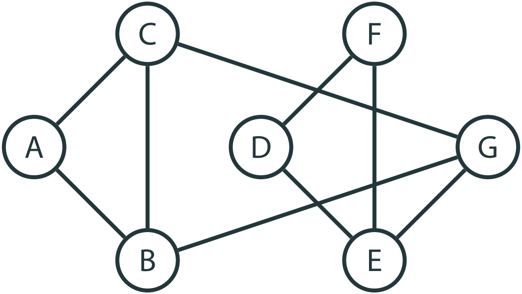
- Важное свойство графа: можно ли из всякой вершины дойти в любую другую?
- Для транспортной задачи говорит о ее разрешимости

# Связность

- Важное свойство графа: можно ли из всякой вершины дойти в любую другую?
- Для транспортной задачи говорит о ее разрешимости
- В целом говорит о наличии связи между частями графа

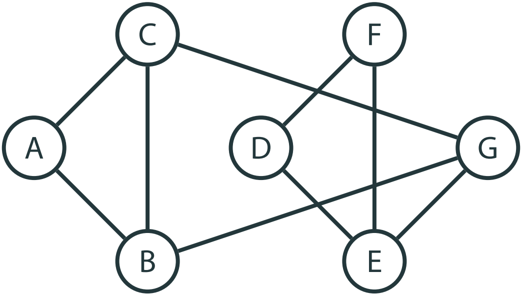
# Связность

- Граф называется связным, если любая вершина достижима из любой другой



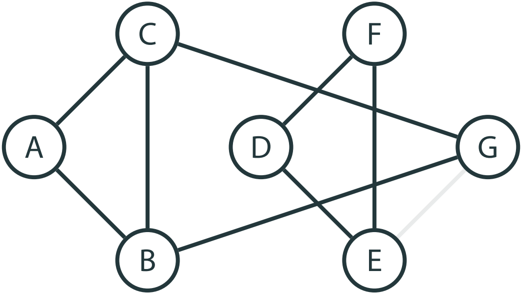
# Связность

- Граф называется связным, если любая вершина достижима из любой другой
- Другими словами, есть путь между любыми двумя ее вершинами



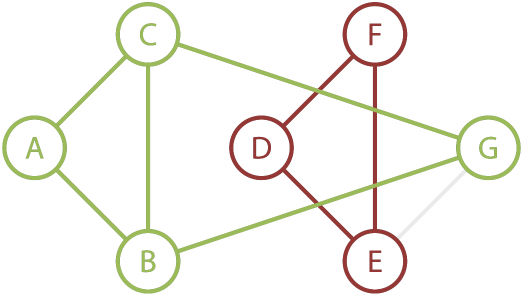
# Связность

- Граф называется связным, если любая вершина достижима из любой другой
- Другими словами, есть путь между любыми двумя ее вершинами
- В противном случае граф не связен



# Связность

- Граф называется **связным**, если любая вершина достижима из любой другой
- Другими словами, есть путь между любыми двумя ее вершинами
- В противном случае граф не связен



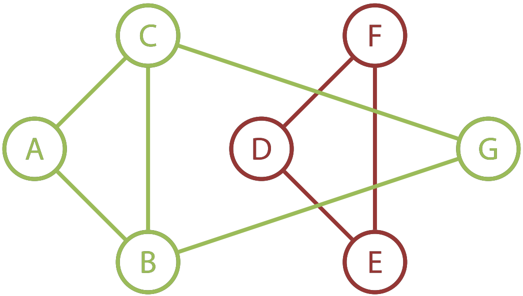
# Компоненты связности

Если граф не связен, все его вершины распадаются на компоненты связности:

# Компоненты связности

Если граф не связен, все его вершины распадаются на компоненты связности:

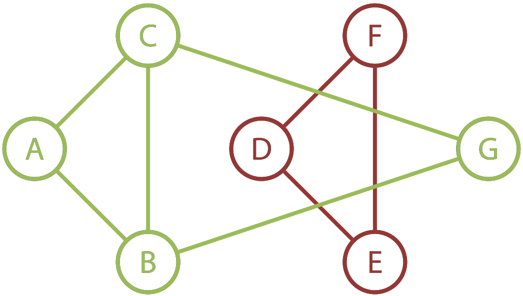
- Каждая вершина лежит ровно в одной компоненте



# Компоненты связности

Если граф не связен, все его вершины распадаются на компоненты связности:

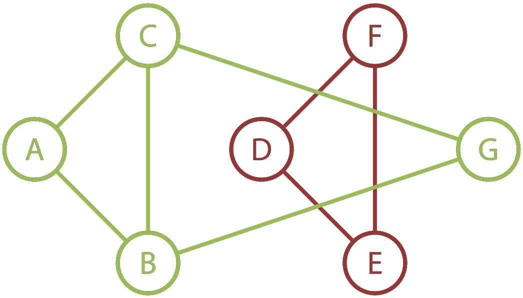
- Каждая вершина лежит ровно в одной компоненте
- Любые вершины в одной компоненте связаны



# Компоненты связности

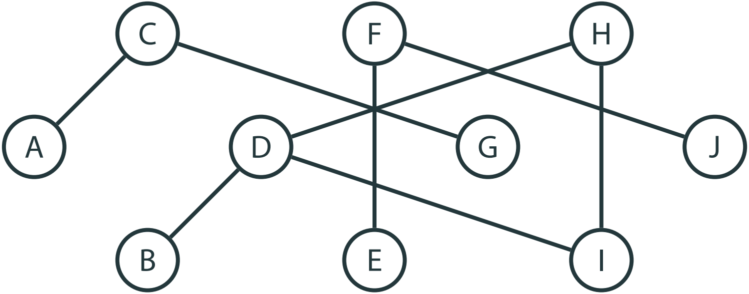
Если граф не связен, все его вершины распадаются на компоненты связности:

- Каждая вершина лежит ровно в одной компоненте
- Любые вершины в одной компоненте связаны
- Вершины из разных компонент не связаны



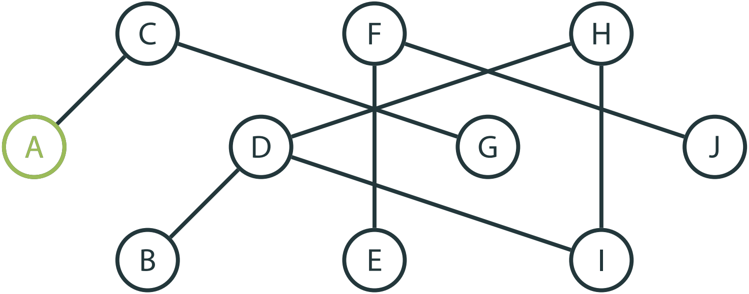
# Как искать компоненты связности?

- Берем любую вершину



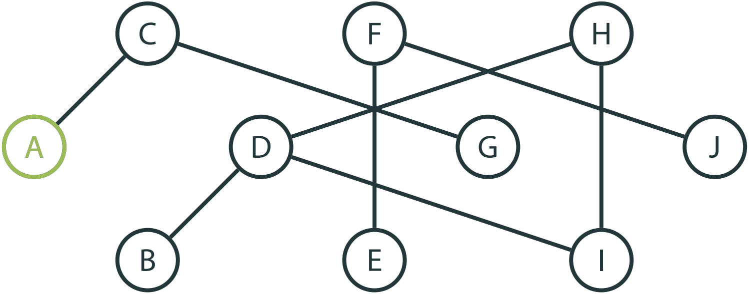
# Как искать компоненты связности?

- Берем любую вершину



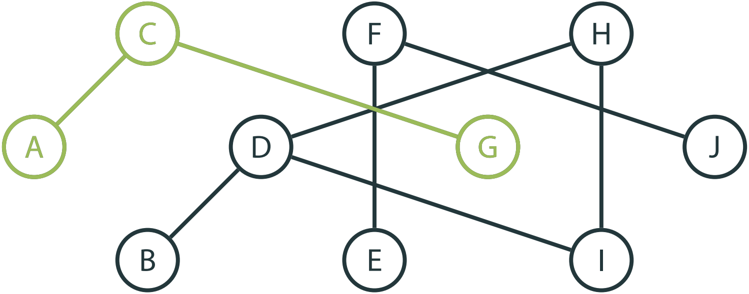
# Как искать компоненты связности?

- Берем любую вершину
- Выделяем все вершины, достижимые из нее



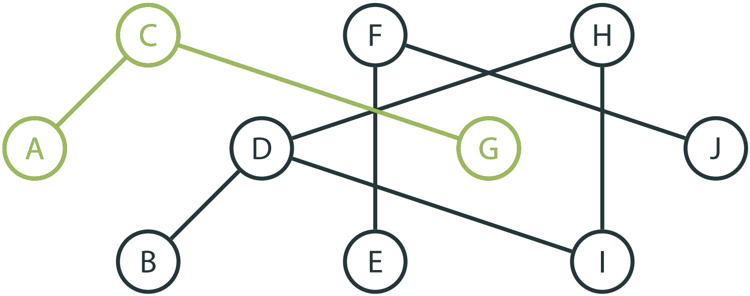
# Как искать компоненты связности?

- Берем любую вершину
- Выделяем все вершины, достижимые из нее



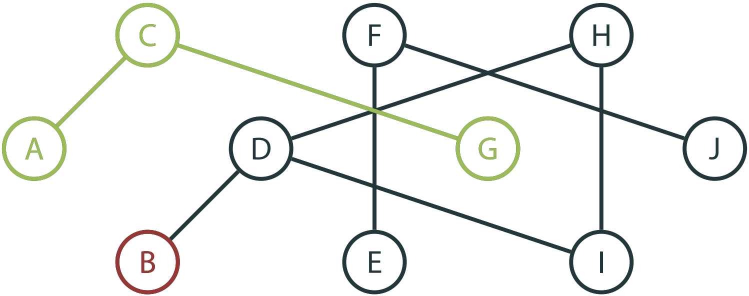
# Как искать компоненты связности?

- Берем любую вершину
- Выделяем все вершины, достижимые из нее
- Повторяем с оставшимися вершинами



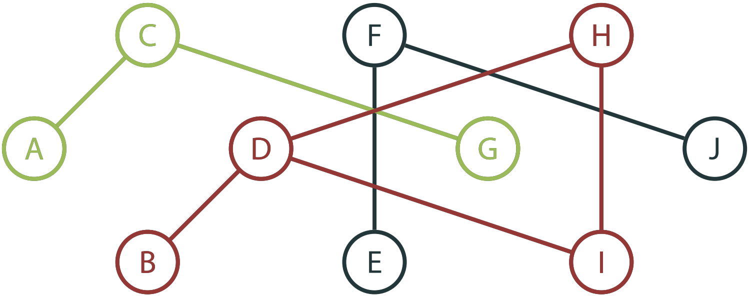
# Как искать компоненты связности?

- Берем любую вершину
- Выделяем все вершины, достижимые из нее
- Повторяем с оставшимися вершинами



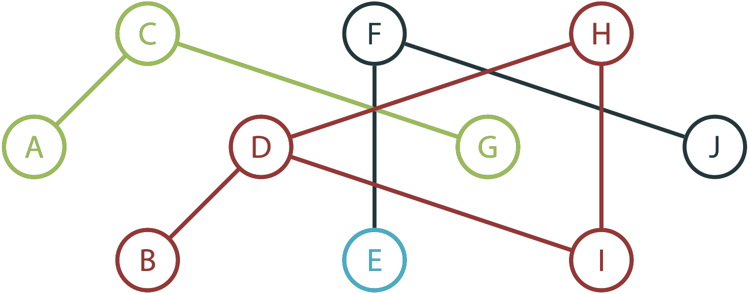
# Как искать компоненты связности?

- Берем любую вершину
- Выделяем все вершины, достижимые из нее
- Повторяем с оставшимися вершинами



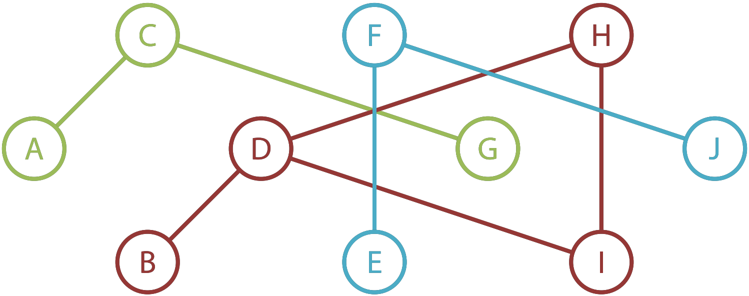
# Как искать компоненты связности?

- Берем любую вершину
- Выделяем все вершины, достижимые из нее
- Повторяем с оставшимися вершинами



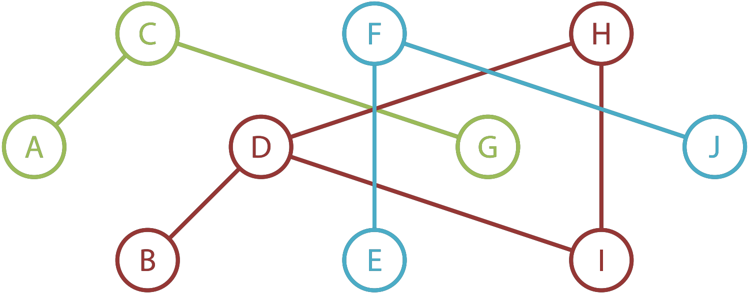
# Как искать компоненты связности?

- Берем любую вершину
- Выделяем все вершины, достижимые из нее
- Повторяем с оставшимися вершинами



# Как искать компоненты связности?

- Берем любую вершину
- Выделяем все вершины, достижимые из нее
- Повторяем с оставшимися вершинами
- Позже обсудим как это делать эффективно



# Пути и достижимость

Пути и достижимость

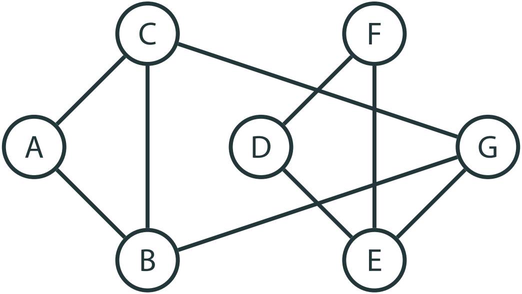
Число компонент связности

Обходы и поиск в графах

Расстояния в графах

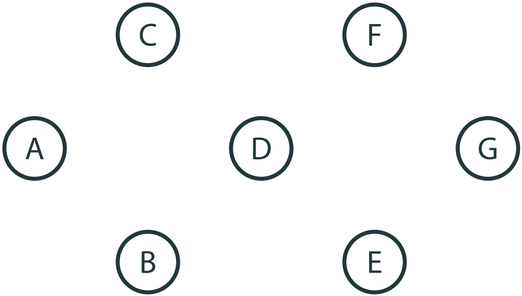
# Число компонент связности

- Число компонент связности может быть от 1 до $|V|$



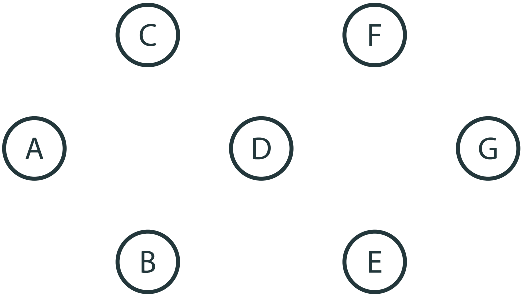
# Число компонент связности

- Число компонент связности может быть от 1 до $|V|$



# Число компонент связности

- Число компонент связности может быть от 1 до $|V|$
- Можно ли сказать что-то более точное, если знать число вершин и число ребер в графе?



# Число компонент связности

**Оценка числа компонент связности**

Число компонент связности в графе не меньше $|V| - |E|$

# Число компонент связности

## Оценка числа компонент связности

Число компонент связности в графе не меньше $|V| - |E|$

- Если $|E| \leq |V| - 2$, то граф не связен

# Число компонент связности

**Оценка числа компонент связности**

Число компонент связности в графе не меньше $|V| - |E|$

- Если $|E| \leq |V| - 2$, то граф не связен
- Если граф связен, то $|E| \geq |V| - 1$

# Число компонент связности

## Оценка числа компонент связности

Число компонент связности в графе не меньше $|V| - |E|$

- Если $|E| \leq |V| - 2$, то граф не связен
- Если граф связен, то $|E| \geq |V| - 1$
- Оценка ничего не говорит, если ребер много ($|E| \geq |V| - 1$)

# Число компонент связности

## Оценка числа компонент связности

Число компонент связности в графе не меньше $|V| - |E|$

- Если $|E| \leq |V| - 2$, то граф не связен
- Если граф связен, то $|E| \geq |V| - 1$
- Оценка ничего не говорит, если ребер много ($|E| \geq |V| - 1$)
- Но при малом числе ребер она полезна

# Число компонент связности

- Докажем оценку

# Число компонент связности

- Докажем оценку
- Выкинем из графа все ребра и будем возвращать их по одному

# Число компонент связности

- Докажем оценку
- Выкинем из графа все ребра и будем возвращать их по одному
- В начале в графе $|V|$ вершин и нет ребер

# Число компонент связности

- Докажем оценку
- Выкинем из графа все ребра и будем возвращать их по одному
- В начале в графе $|V|$ вершин и нет ребер
- Число компонент связности равно $|V|$ и оценка верна: $|V| \geq |V| - 0$

# Число компонент связности

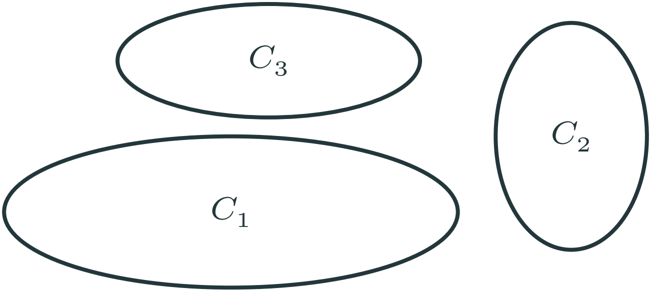
- Докажем оценку
- Выкинем из графа все ребра и будем возвращать их по одному
- В начале в графе $|V|$ вершин и нет ребер
- Число компонент связности равно $|V|$ и оценка верна: $|V| \geq |V| - 0$
- При возвращении одного ребра величина $|V| - |E|$ уменьшается на 1

# Число компонент связности

- Докажем оценку
- Выкинем из графа все ребра и будем возвращать их по одному
- В начале в графе $|V|$ вершин и нет ребер
- Число компонент связности равно $|V|$ и оценка верна: $|V| \geq |V| - 0$
- При возвращении одного ребра величина $|V| - |E|$ уменьшается на 1
- Посмотрим, что происходит с числом компонент связности

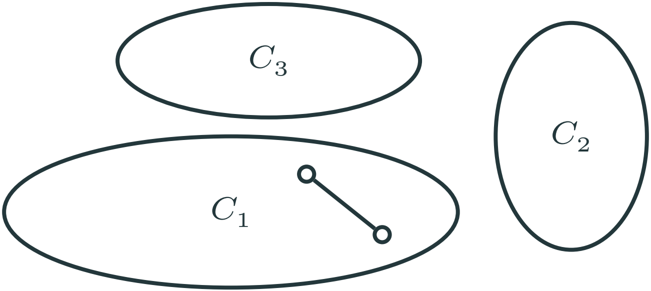
# Число компонент связности

- Выделим текущие компоненты связности



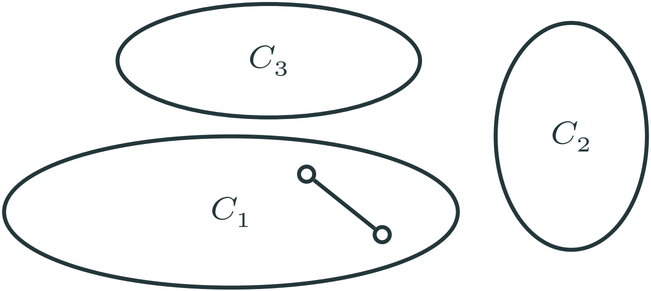
# Число компонент связности

- Выделим текущие компоненты связности
- Случай 1: ребро соединяет вершины в одной компоненте



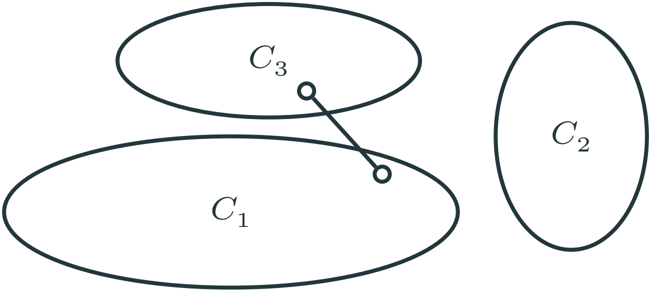
# Число компонент связности

- Выделим текущие компоненты связности
- Случай 1: ребро соединяет вершины в одной компоненте
- Тогда компоненты остаются те же



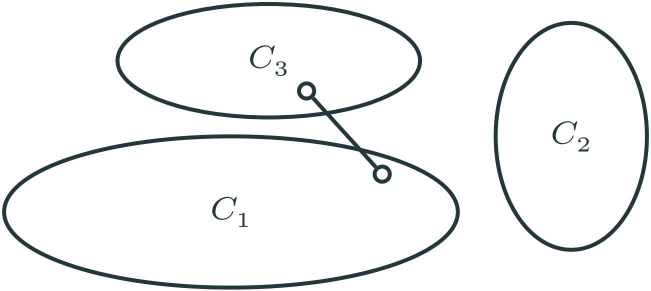
# Число компонент связности

- Выделим текущие компоненты связности
- Случай 2: ребро соединяет вершины в разных компонентах



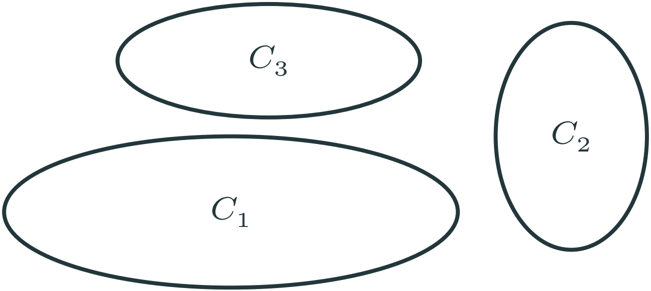
# Число компонент связности

- Выделим текущие компоненты связности
- Случай 2: ребро соединяет вершины в разных компонентах
- Тогда две компоненты сливаются в одну



# Число компонент связности

- При возвращении одного ребра число компонент связности либо не меняется, либо уменьшается на 1



# Число компонент связности

- Итак, в начале число компонент связности не меньше $|V| - |E|$

# Число компонент связности

- Итак, в начале число компонент связности не меньше $|V| - |E|$
- При возвращении ребра число компонент связности может не измениться, а может уменьшиться на 1

# Число компонент связности

- Итак, в начале число компонент связности не меньше $|V| - |E|$
- При возвращении ребра число компонент связности может не измениться, а может уменьшиться на 1
- Величина $|V| - |E|$ точно уменьшается на один при возвращении ребра

# Число компонент связности

- Итак, в начале число компонент связности не меньше $|V| - |E|$
- При возвращении ребра число компонент связности может не измениться, а может уменьшиться на 1
- Величина $|V| - |E|$ точно уменьшается на один при возвращении ребра
- Значит после возвращения ребра неравенство остается верным!

# Число компонент связности

- Итак, в начале число компонент связности не меньше $|V| - |E|$
- При возвращении ребра число компонент связности может не измениться, а может уменьшиться на 1
- Величина $|V| - |E|$ точно уменьшается на один при возвращении ребра
- Значит после возвращения ребра неравенство остается верным!
- Значит оно останется верным после возвращения всех ребер!

# Самый тяжелый камень

**Самый тяжелый камень**

У нас есть $n$ камней и чашечные весы. За одно взвешивание мы можем сравнить по весу два камня. Сколько нужно взвешиваний, чтобы гарантировано найти самый тяжелый камень?

# Самый тяжелый камень

### Самый тяжелый камень

У нас есть $n$ камней и чашечные весы. За одно взвешивание мы можем сравнить по весу два камня. Сколько нужно взвешиваний, чтобы гарантировано найти самый тяжелый камень?

- Сначала не вполне ясно, причем тут графы и компоненты связности

# Самый тяжелый камень

**Самый тяжелый камень**

У нас есть $n$ камней и чашечные весы. За одно взвешивание мы можем сравнить по весу два камня. Сколько нужно взвешиваний, чтобы гарантировано найти самый тяжелый камень?

- Сначала не вполне ясно, причем тут графы и компоненты связности
- Но давайте разбираться

# Самый тяжелый камень

- Легко понять, что $n-1$ взвешивания хватит

# Самый тяжелый камень

- Легко понять, что $n-1$ взвешивания хватит
- После каждого взвешивания мы можем отбрасывать более легкий камень

# Самый тяжелый камень

- Легко понять, что $n-1$ взвешивания хватит
- После каждого взвешивания мы можем отбрасывать более легкий камень
- После $n-1$ взвешивания у нас останется один камень, он будет самым тяжелым

# Самый тяжелый камень

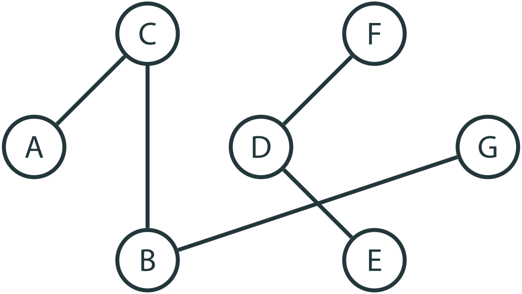
- Легко понять, что $n-1$ взвешивания хватит
- После каждого взвешивания мы можем отбрасывать более легкий камень
- После $n-1$ взвешивания у нас останется один камень, он будет самым тяжелым
- Но можно ли обойтись меньшем числом взвешиваний?

# Самый тяжелый камень

- Легко понять, что $n-1$ взвешивания хватит
- После каждого взвешивания мы можем отбрасывать более легкий камень
- После $n-1$ взвешивания у нас останется один камень, он будет самым тяжелым
- Но можно ли обойтись меньшем числом взвешиваний?
- Оказывается, нет!

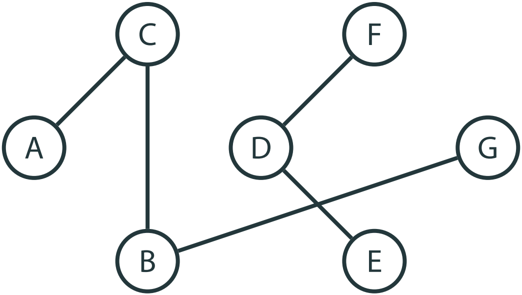
# Самый тяжелый камень

- Давайте рассмотрим такой граф: вершинами являются камни, а ребрами мы соединяем те камни, которые мы сравнивали на весах



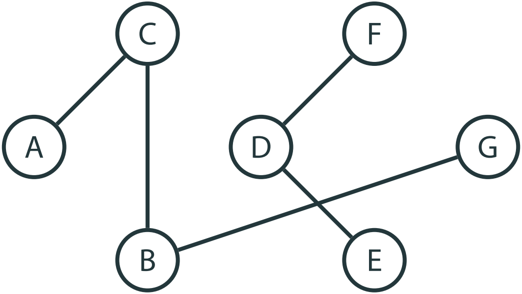
# Самый тяжелый камень

- Давайте рассмотрим такой граф: вершинами являются камни, а ребрами мы соединяем те камни, которые мы сравнивали на весах
- Заметим, что мы даже не интересуемся результатом взвешиваний



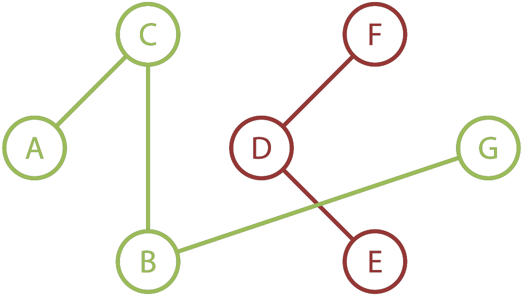
# Самый тяжелый камень

- Если мы сделали меньше $n - 1$ взвешивания, то в нашем графе не меньше двух компонент связности!



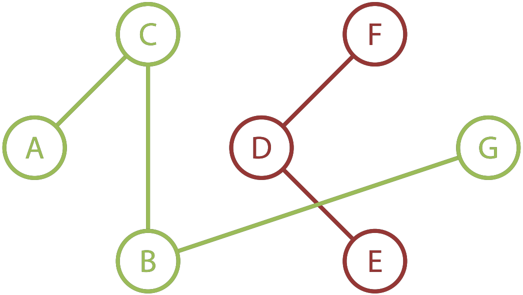
# Самый тяжелый камень

- Если мы сделали меньше $n-1$ взвешивания, то в нашем графе не меньше двух компонент связности!



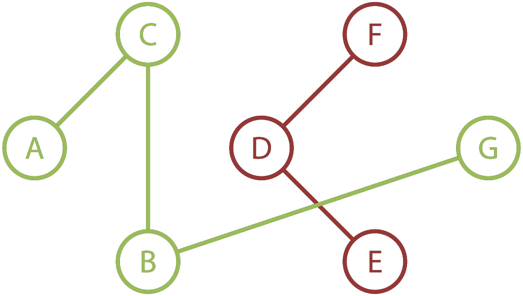
# Самый тяжелый камень

- Если мы сделали меньше $n - 1$ взвешивания, то в нашем графе не меньше двух компонент связности!
- Значит мы не сравнивали камни двух этих компонент друг с другом



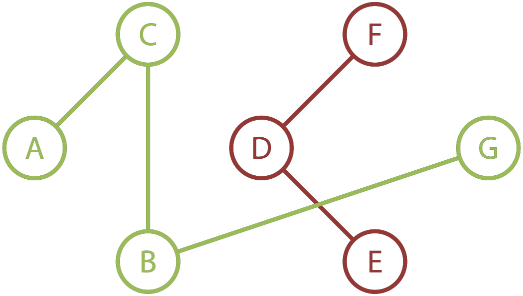
# Самый тяжелый камень

- Все камни в одной компоненте могут быть сильно тяжелее всех камней в другой компоненте, или наоборот



# Самый тяжелый камень

- Все камни в одной компоненте могут быть сильно тяжелее всех камней в другой компоненте, или наоборот
- Значит, мы не знаем какой камень самый тяжелый



# Пути и достижимость

Пути и достижимость

Число компонент связности

Обходы и поиск в графах

Расстояния 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

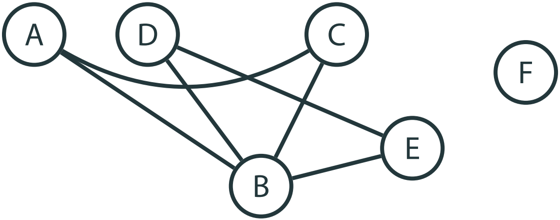
- Это можно описать рекурсивно

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

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

# Поиск в глубину

- Это можно описать рекурсивно

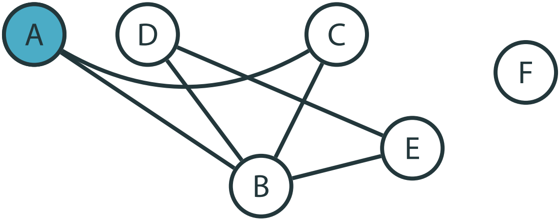
```
def Explore(v):
    visited[v]=True

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



# Поиск в глубину

- Это можно описать рекурсивно

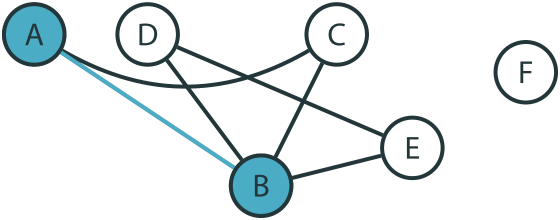
```
def Explore(v):
    visited[v]=True

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



# Поиск в глубину

- Это можно описать рекурсивно

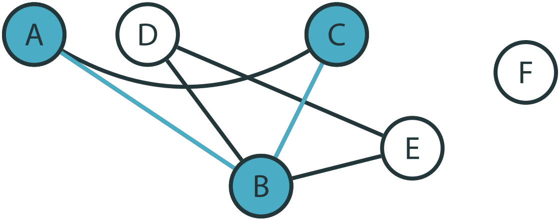
```
def Explore(v):
    visited[v]=True

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



# Поиск в глубину

- Это можно описать рекурсивно

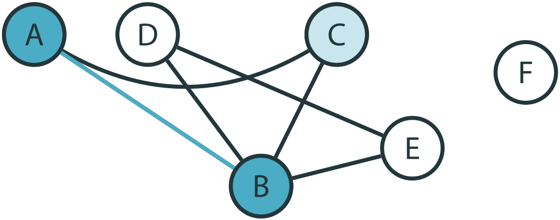
```
def Explore(v):
    visited[v]=True

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



# Поиск в глубину

- Это можно описать рекурсивно

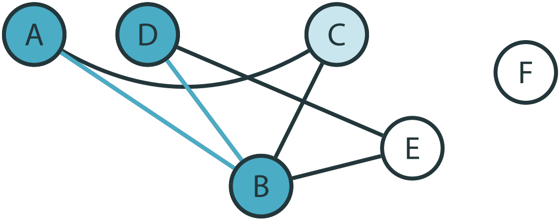
```
def Explore(v):
    visited[v]=True

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



# Поиск в глубину

- Это можно описать рекурсивно

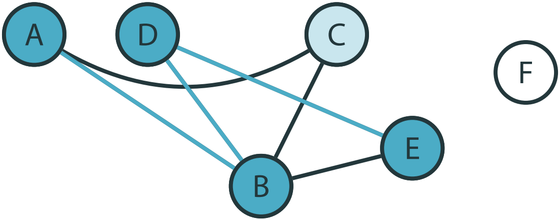
```
def Explore(v):
    visited[v]=True

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



# Поиск в глубину

- Это можно описать рекурсивно

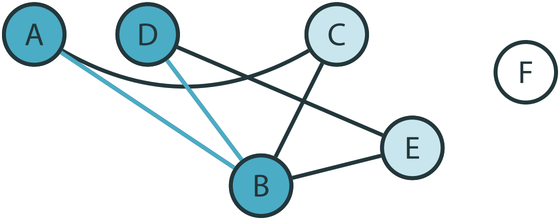
```
def Explore(v):
    visited[v]=True

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



# Поиск в глубину

- Это можно описать рекурсивно

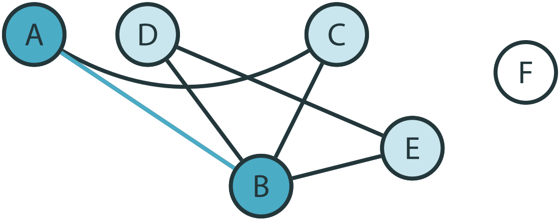
```
def Explore(v):
    visited[v]=True

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



# Поиск в глубину

- Это можно описать рекурсивно

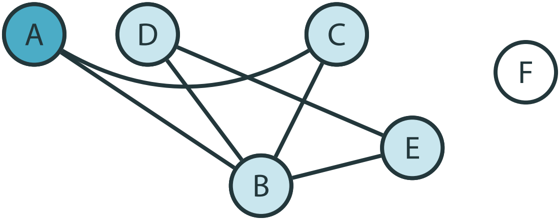
```
def Explore(v):
    visited[v]=True

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



# Поиск в глубину

- Это можно описать рекурсивно

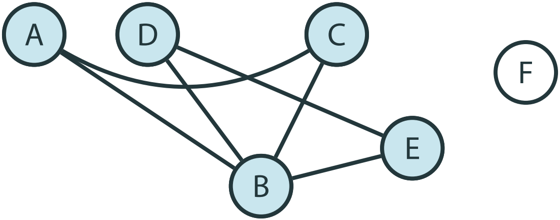
```
def Explore(v):
    visited[v]=True

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



# Поиск в глубину

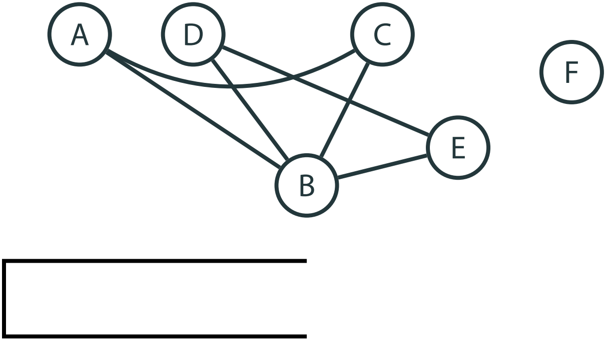
- Это можно описать рекурсивно

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

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



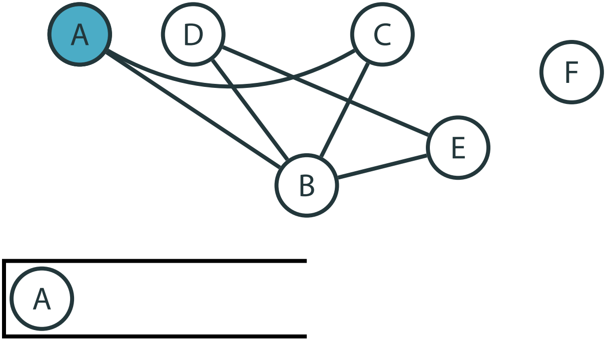
# Поиск в глубину

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



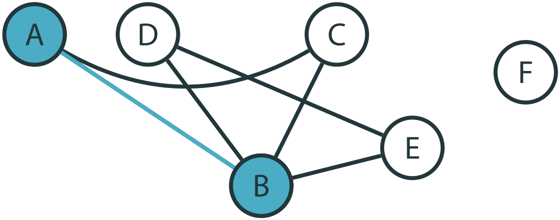
# Поиск в глубину

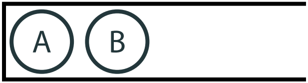
По существу, у нас поддерживается стек рассматриваемых в данный момент вершин



# Поиск в глубину

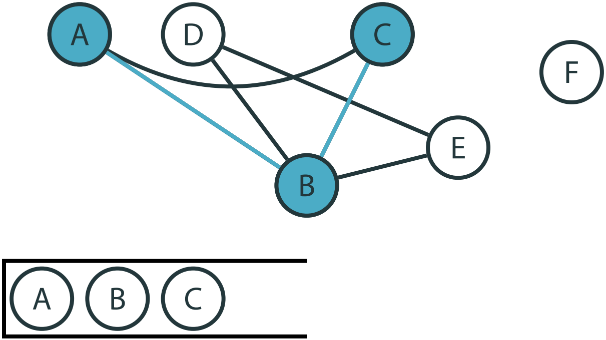
По существу, у нас поддерживается стек рассматриваемых в данный момент вершин



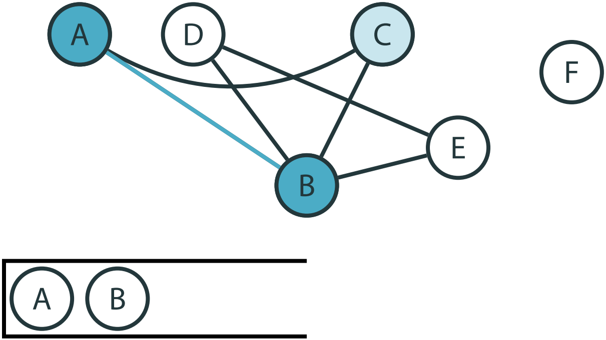
# Поиск в глубину

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



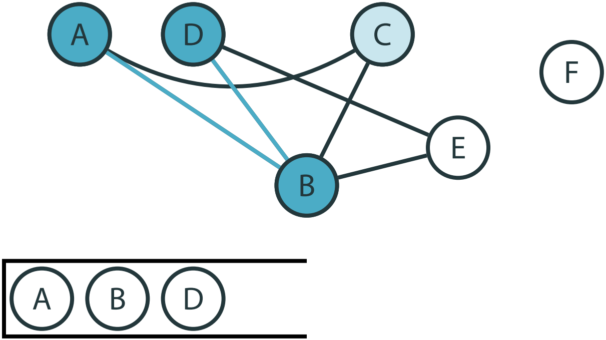
# Поиск в глубину

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



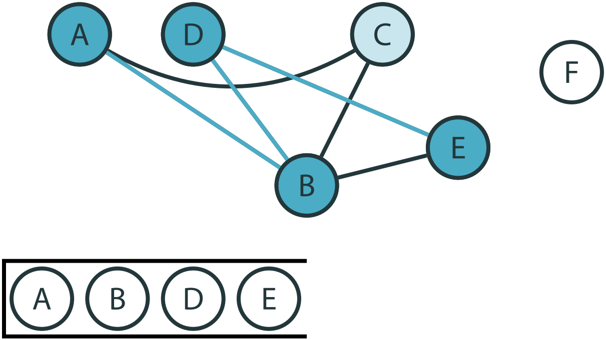
# Поиск в глубину

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



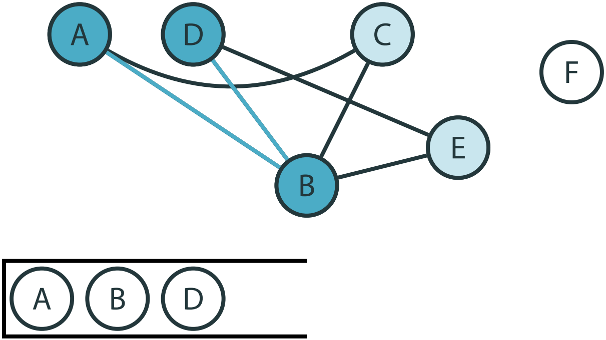
# Поиск в глубину

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



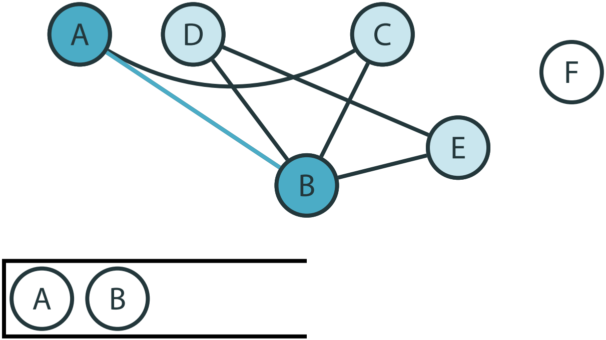
# Поиск в глубину

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



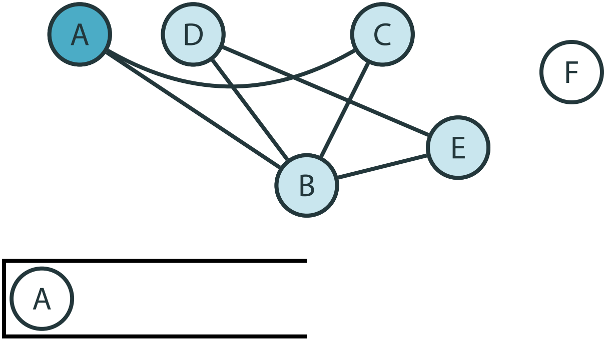
# Поиск в глубину

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



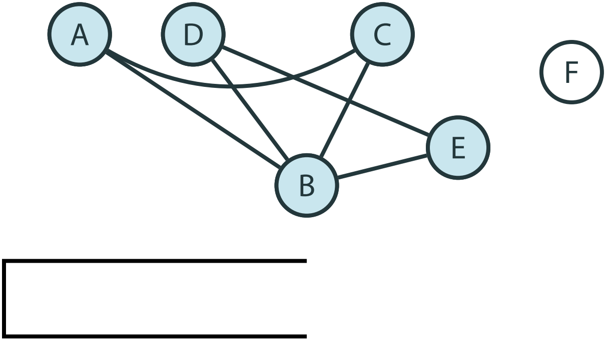
# Поиск в глубину

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

- 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

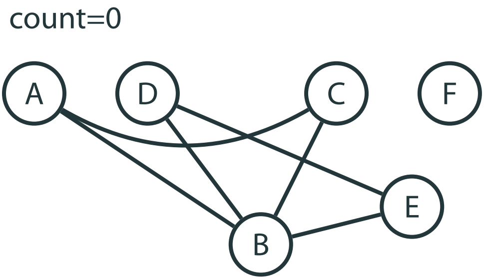
```
def Previsit(v):
    pre[v]=clock
    clock+=1

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
```

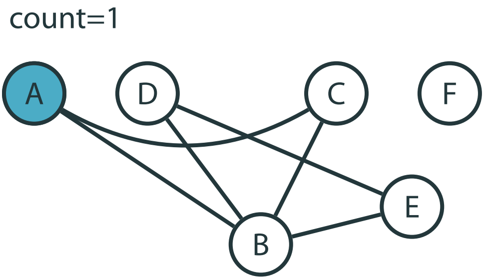
# Поиск в глубину




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

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

| | pre | post |
|---|---|---|
| A | | |
| B | | |
| C | | |
| D | | |
| E | | |
| F | | |

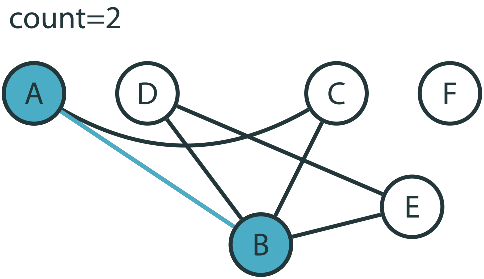
# Поиск в глубину




```
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

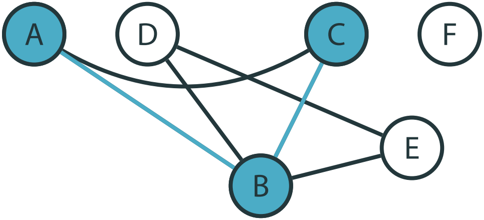



```
def Previsit(v):
    pre[v]=clock
    clock+=1

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

| | pre | post |
|---|---|---|
| A | 0 | |
| B | 1 | |
| C | | |
| D | | |
| E | | |
| F | | |

# Поиск в глубину

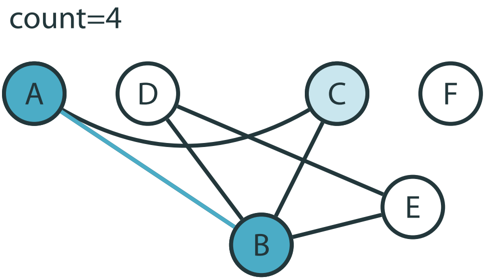
count=3




```
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

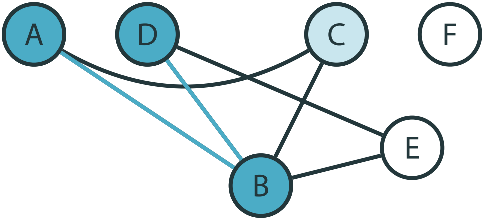



```
def  Previsit(v):
     pre[v]=clock
     clock+=1

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

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

# Поиск в глубину

count=5

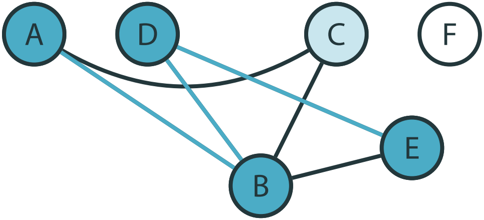



```
def Previsit(v):
    pre[v]=clock
    clock+=1

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

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

# Поиск в глубину

count=6

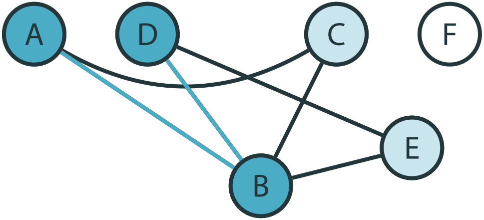



```
def  Previsit(v):
     pre[v]=clock
     clock+=1

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

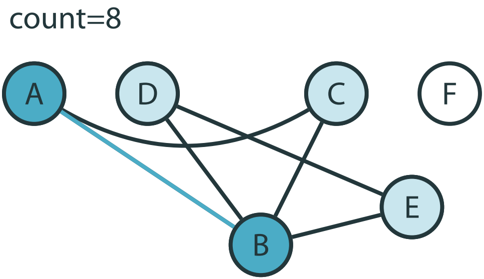
count=7




```
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

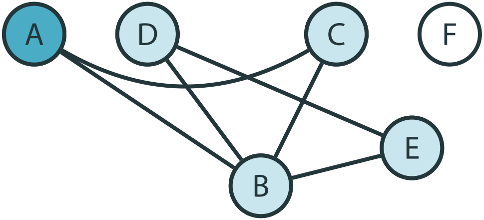



```
def Previsit(v):
    pre[v]=clock
    clock+=1

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

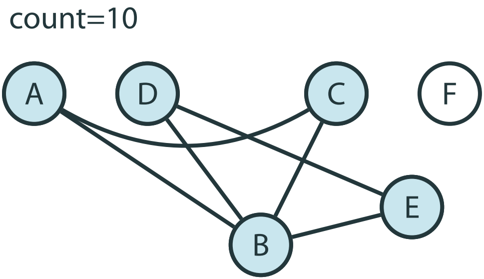
count=9




```
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

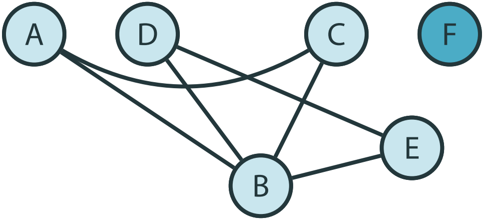



```
def Previsit(v):
    pre[v]=clock
    clock+=1

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

count=11

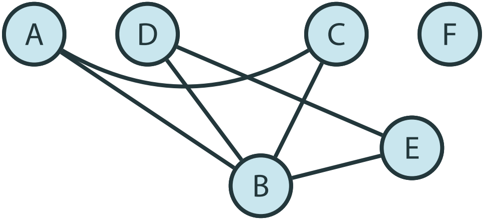



```
def  Previsit(v):
     pre[v]=clock
     clock+=1

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

count=12

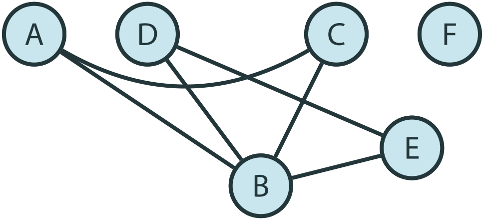



```
def Previsit(v):
    pre[v]=clock
    clock+=1

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

count=12




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

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

# 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
```

# Пути и достижимость

Пути и достижимость

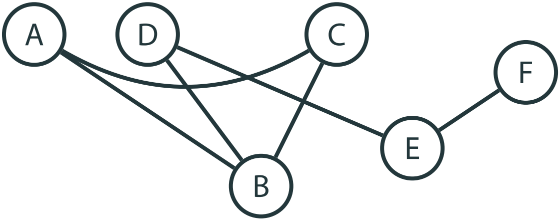
Число компонент связности

Обходы и поиск в графах

Расстояния в графах

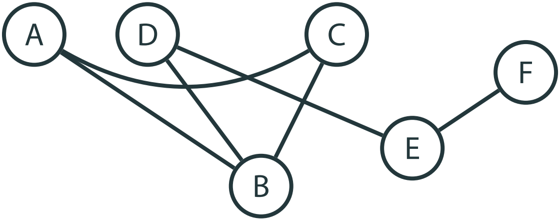
# Расстояния

- Рассмотрим граф



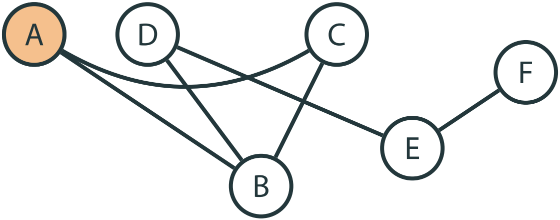
# Расстояния

- Рассмотрим граф
- Расстоянием между вершинами в графе называется длина кратчайшего пути между ними



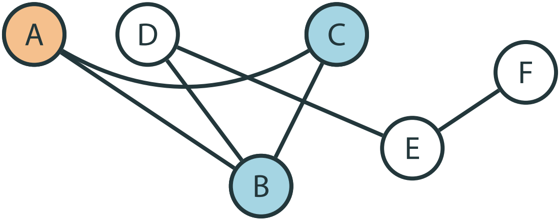
# Расстояния

- Рассмотрим граф
- Расстоянием между вершинами в графе называется длина кратчайшего пути между ними
- Зафиксируем вершину $A$



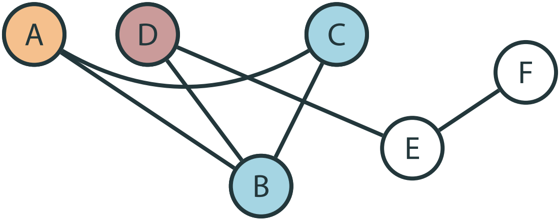
# Расстояния

- Ее соседи на расстоянии 1



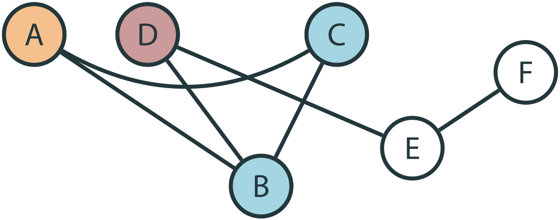
# Расстояния

- Ее соседи на расстоянии 1
- Соседи соседей на расстоянии 2



# Расстояния

- Ее соседи на расстоянии 1
- Соседи соседей на расстоянии 2
- И так далее



# Расстояния

- В соцсетях расстояния — длина минимальной цепочка знакомств между людьми

# Расстояния

- В соцсетях расстояния — длина минимальной цепочка знакомств между людьми
- В транспортных графах расстояния носят естественный смысл

# Расстояния

- В соцсетях расстояния — длина минимальной цепочка знакомств между людьми
- В транспортных графах расстояния носят естественный смысл
- Важно искать кратчайшие расстояния

# Расстояния

- В соцсетях расстояния — длина минимальной цепочка знакомств между людьми
- В транспортных графах расстояния носят естественный смысл
- Важно искать кратчайшие расстояния
- Важно искать близкие вершины

# Нахождение расстояний

- Как вычислить расстояние от вершины до всех остальных в графе?

# Нахождение расстояний

- Как вычислить расстояние от вершины до всех остальных в графе?
- Поиск в ширину

# Нахождение расстояний

- Как вычислить расстояние от вершины до всех остальных в графе?
- Поиск в ширину
- Начинаем с самой вершины

# Нахождение расстояний

- Как вычислить расстояние от вершины до всех остальных в графе?
- Поиск в ширину
- Начинаем с самой вершины
- Добавляем вершины на расстоянии 1

# Нахождение расстояний

- Как вычислить расстояние от вершины до всех остальных в графе?
- Поиск в ширину
- Начинаем с самой вершины
- Добавляем вершины на расстоянии 1
- Перебираем их и добавляем вершины на расстоянии 2

# Нахождение расстояний

- Как вычислить расстояние от вершины до всех остальных в графе?
- Поиск в ширину
- Начинаем с самой вершины
- Добавляем вершины на расстоянии 1
- Перебираем их и добавляем вершины на расстоянии 2
- Перебираем их и добавляем вершины на расстоянии 3

# Нахождение расстояний

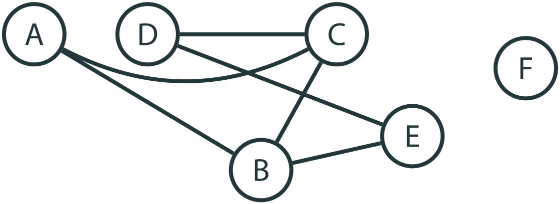
- Как вычислить расстояние от вершины до всех остальных в графе?
- Поиск в ширину
- Начинаем с самой вершины
- Добавляем вершины на расстоянии 1
- Перебираем их и добавляем вершины на расстоянии 2
- Перебираем их и добавляем вершины на расстоянии 3
- И так далее

# Реализация

```
from collections import deque

def bfs(G, v):
  dist = {v: 0}
  queue = deque([v])

  while queue:
    s = queue.popleft()
    for u in G[s]:
      if u not in dist:
        queue.append(u)
        dist[u]=dist[s]+1
  return dist
```

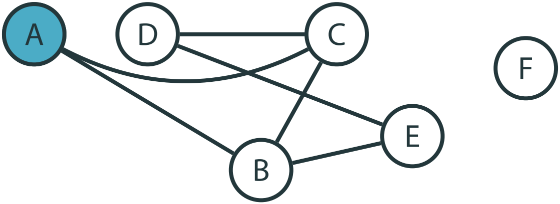
# Реализация

```
from collections import deque

def bfs(G, v):
  dist = {v: 0}
  queue = deque([v])

  while queue:
    s = queue.popleft()
    for u in G[s]:
      if u not in dist:
        queue.append(u)
        dist[u]=dist[s]+1
  return dist
```



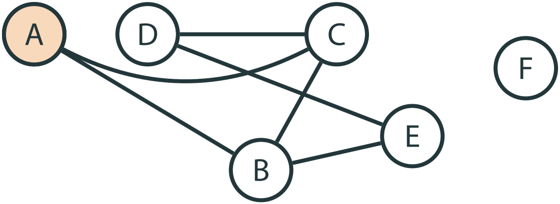
# Реализация

```
from collections import deque

def bfs(G, v):
  dist = {v: 0}
  queue = deque([v])

  while queue:
    s = queue.popleft()
    for u in G[s]:
      if u not in dist:
        queue.append(u)
        dist[u]=dist[s]+1
  return dist
```



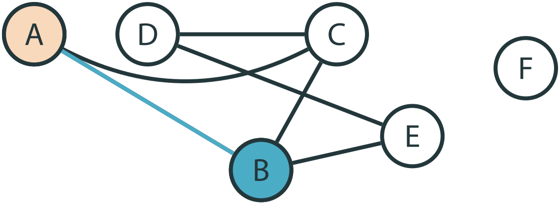
# Реализация

```
from collections import deque

def bfs(G, v):
  dist = {v: 0}
  queue = deque([v])

  while queue:
    s = queue.popleft()
    for u in G[s]:
      if u not in dist:
        queue.append(u)
        dist[u]=dist[s]+1
  return dist
```



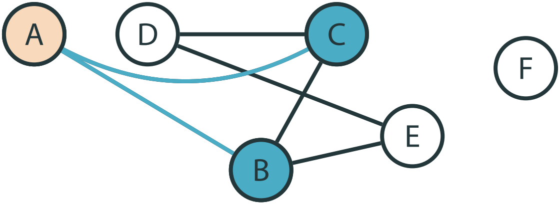
# Реализация

```
from collections import deque

def bfs(G, v):
  dist = {v: 0}
  queue = deque([v])

  while queue:
    s = queue.popleft()
    for u in G[s]:
      if u not in dist:
        queue.append(u)
        dist[u]=dist[s]+1
  return dist
```



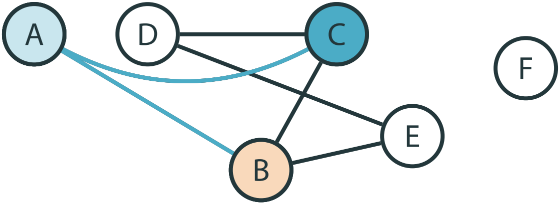
# Реализация

```python
from collections import deque

def bfs(G, v):
  dist = {v: 0}
  queue = deque([v])

  while queue:
    s = queue.popleft()
    for u in G[s]:
      if u not in dist:
        queue.append(u)
        dist[u]=dist[s]+1
  return dist
```



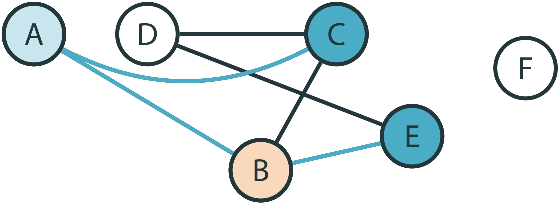
# Реализация

```
from collections import deque

def bfs(G, v):
  dist = {v: 0}
  queue = deque([v])

  while queue:
    s = queue.popleft()
    for u in G[s]:
      if u not in dist:
        queue.append(u)
        dist[u]=dist[s]+1
  return dist
```



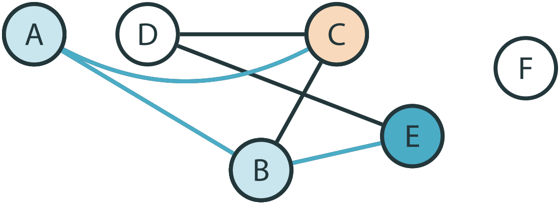
# Реализация

```
from collections import deque

def bfs(G, v):
  dist = {v: 0}
  queue = deque([v])

  while queue:
    s = queue.popleft()
    for u in G[s]:
      if u not in dist:
        queue.append(u)
        dist[u]=dist[s]+1
  return dist
```



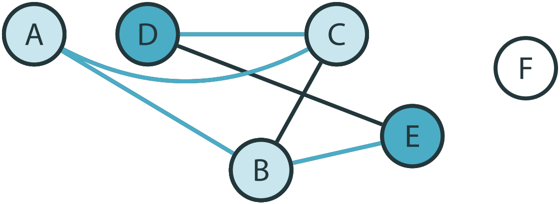
# Реализация

```
from collections import deque

def bfs(G, v):
  dist = {v: 0}
  queue = deque([v])

  while queue:
    s = queue.popleft()
    for u in G[s]:
      if u not in dist:
        queue.append(u)
        dist[u]=dist[s]+1
  return dist
```



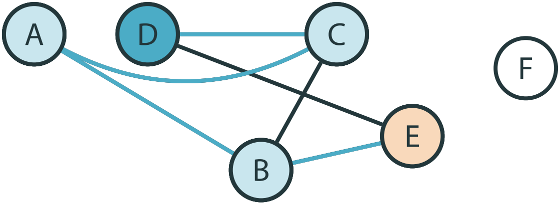
# Реализация

```
from collections import deque

def bfs(G, v):
  dist = {v: 0}
  queue = deque([v])

  while queue:
    s = queue.popleft()
    for u in G[s]:
      if u not in dist:
        queue.append(u)
        dist[u]=dist[s]+1
  return dist
```



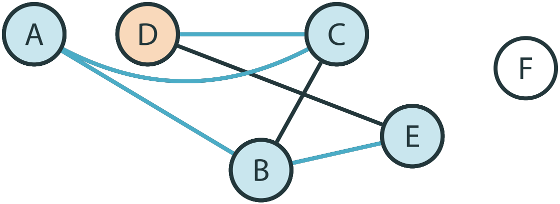
# Реализация

```python
from collections import deque

def bfs(G, v):
  dist = {v: 0}
  queue = deque([v])

  while queue:
    s = queue.popleft()
    for u in G[s]:
      if u not in dist:
        queue.append(u)
        dist[u]=dist[s]+1
  return dist
```



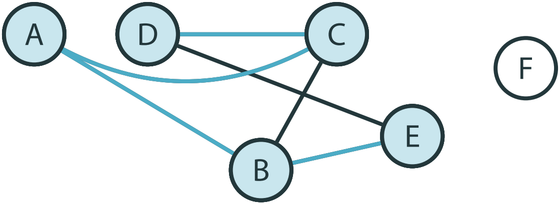
# Реализация

```
from collections import deque

def bfs(G, v):
  dist = {v: 0}
  queue = deque([v])

  while queue:
    s = queue.popleft()
    for u in G[s]:
      if u not in dist:
        queue.append(u)
        dist[u]=dist[s]+1
  return dist
```



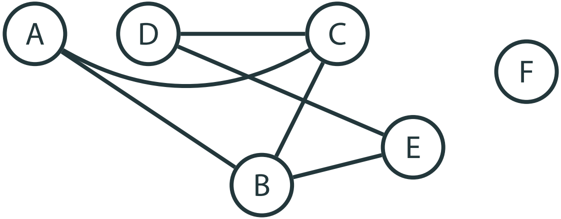
# Реализация

```python
from collections import deque

def bfs(G, v):
  dist = {v: 0}
  queue = deque([v])

  while queue:
    s = queue.popleft()
    for u in G[s]:
      if u not in dist:
        queue.append(u)
        dist[u]=dist[s]+1
  return dist
```



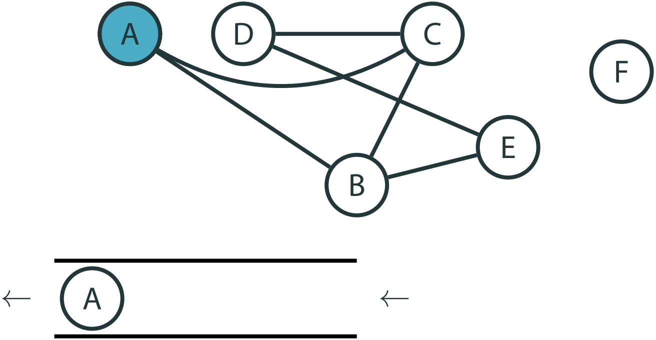
# Поиск в ширину

- Поддерживаем очередь рассматриваемых в данный момент вершин



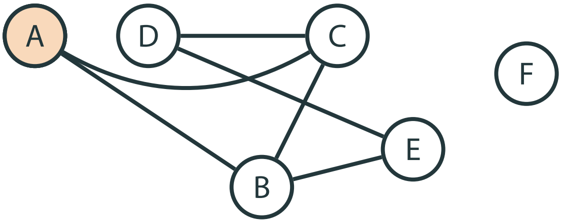
# Поиск в ширину

- Поддерживаем очередь рассматриваемых в данный момент вершин



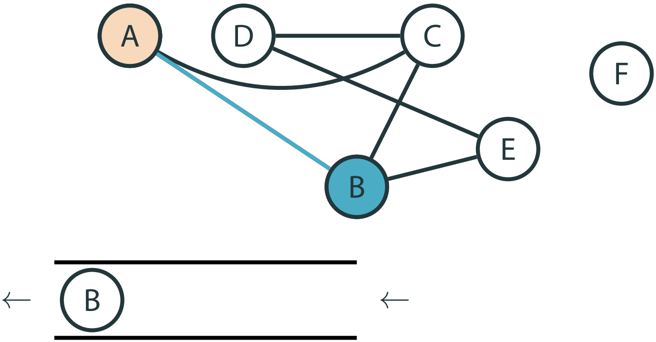
# Поиск в ширину

- Поддерживаем очередь рассматриваемых в данный момент вершин



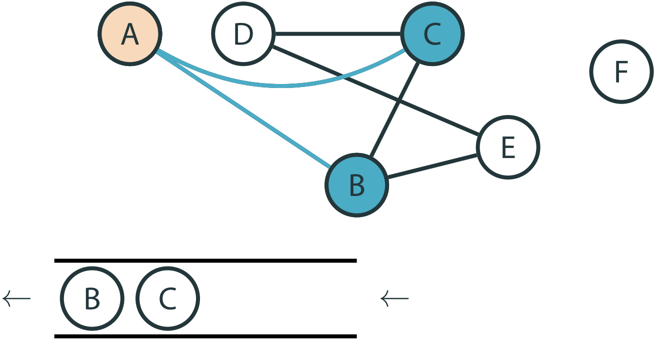

# Поиск в ширину

- Поддерживаем очередь рассматриваемых в данный момент вершин



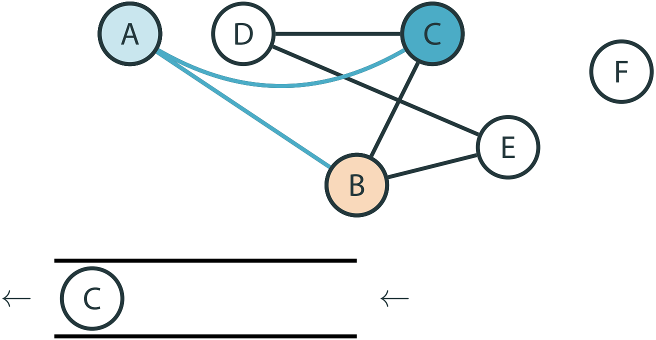
# Поиск в ширину

- Поддерживаем очередь рассматриваемых в данный момент вершин



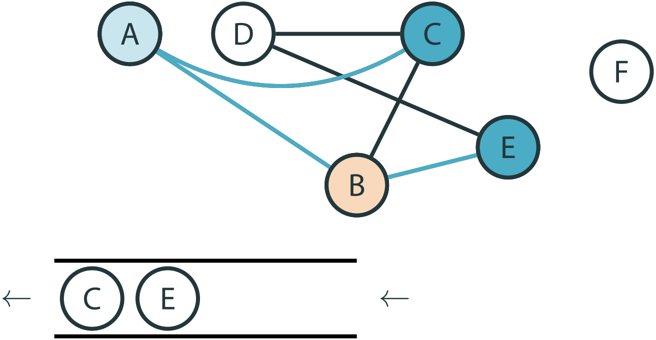
# Поиск в ширину

- Поддерживаем очередь рассматриваемых в данный момент вершин



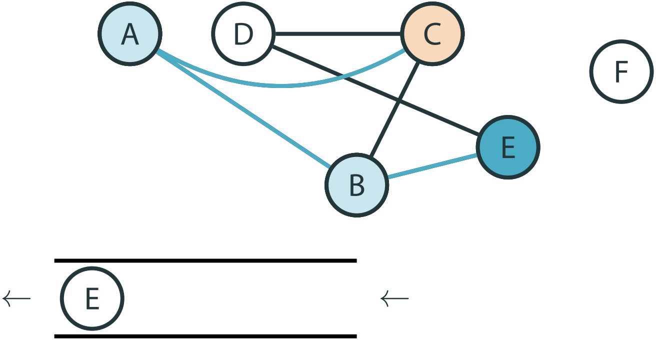
# Поиск в ширину

- Поддерживаем очередь рассматриваемых в данный момент вершин



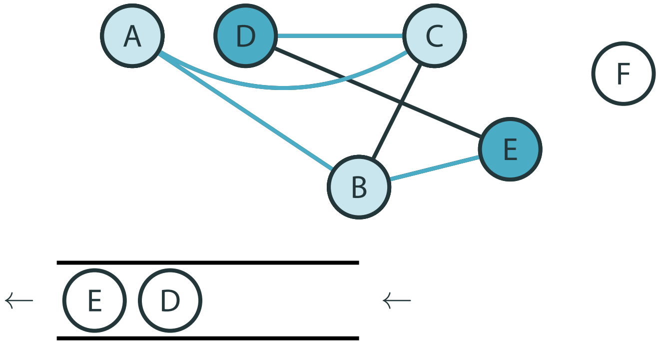
# Поиск в ширину

- Поддерживаем очередь рассматриваемых в данный момент вершин



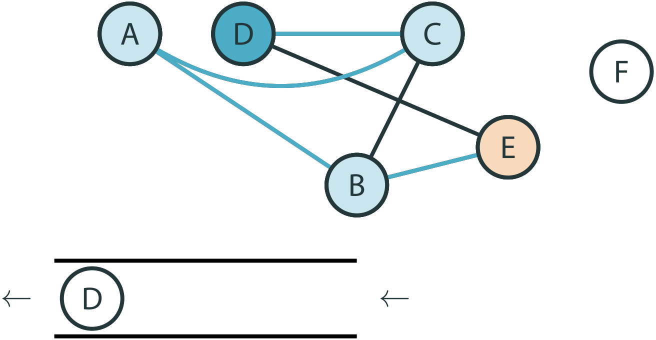
# Поиск в ширину

- Поддерживаем очередь рассматриваемых в данный момент вершин



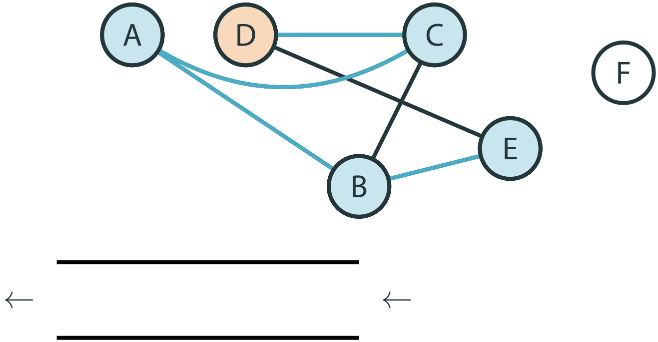
# Поиск в ширину

- Поддерживаем очередь рассматриваемых в данный момент вершин



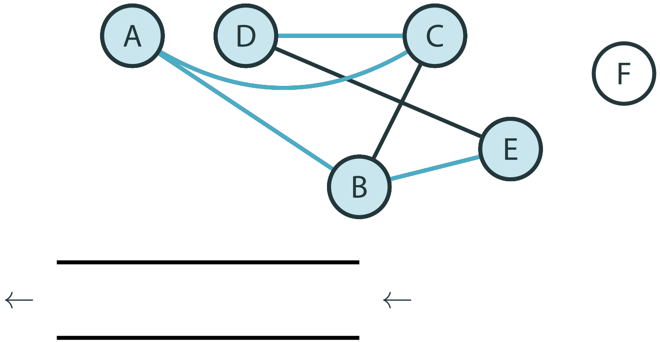
# Поиск в ширину

- Поддерживаем очередь рассматриваемых в данный момент вершин



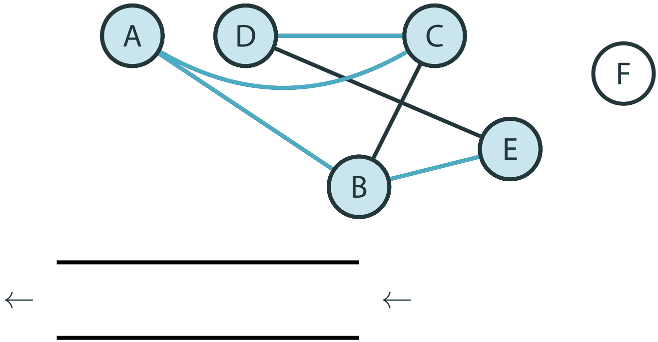
# Поиск в ширину

- Поддерживаем очередь рассматриваемых в данный момент вершин



# Поиск в ширину

- Поддерживаем очередь рассматриваемых в данный момент вершин
- Отличие от поиска в глубину по существу в использовании очереди вместо стека



# Что мы узнали

- Пути играют важную роль при изучении графов

# Что мы узнали

- Пути играют важную роль при изучении графов
- Связность — важное свойства графа

# Что мы узнали

- Пути играют важную роль при изучении графов
- Связность — важное свойства графа
- Поиски в глубину и ширину можно использовать для обхода графа

# Что мы узнали

- Пути играют важную роль при изучении графов
- Связность — важное свойства графа
- Поиски в глубину и ширину можно использовать для обхода графа
- Какой из них лучше использовать зависит от задачи