Графы – 1. Введение.

Шовкопляс Григорий

Введение в алгоритмы и структуры данных

Что такое графы

Графы

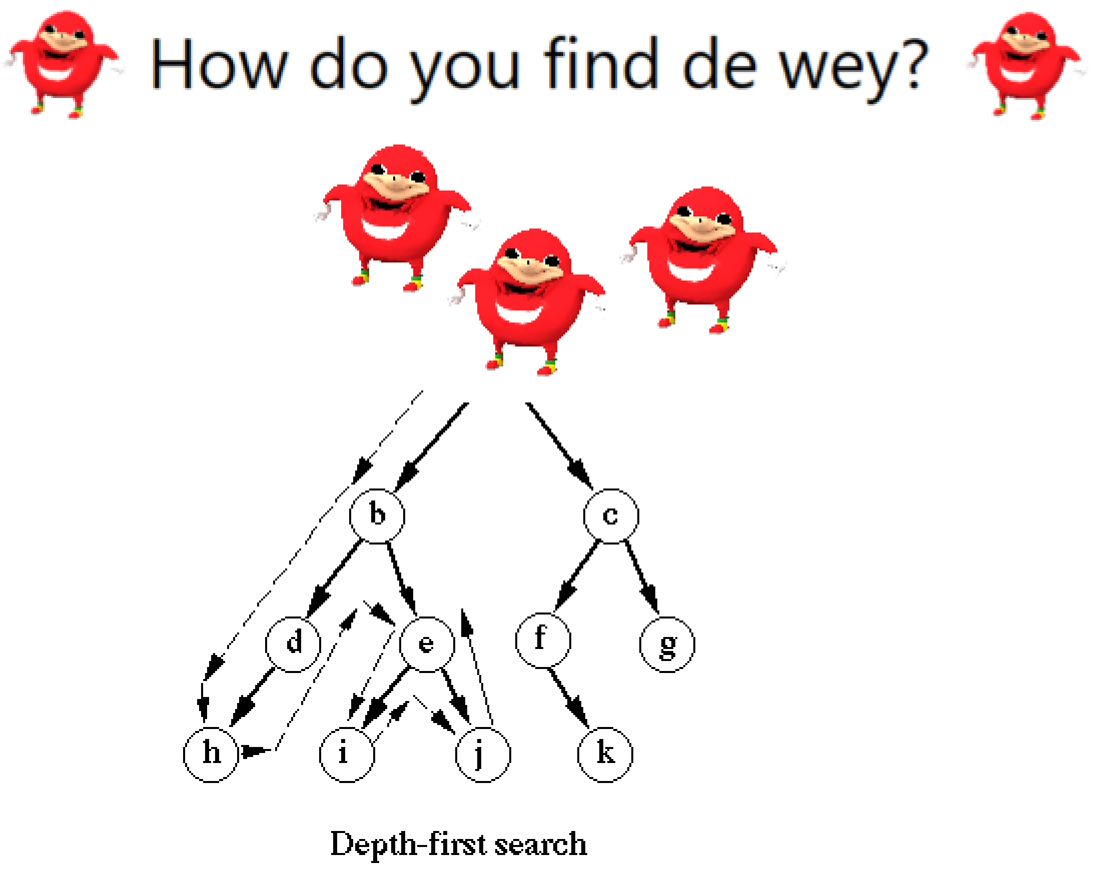
- Граф: G = <V, E>
  - V – множество вершин
  - E – множество ребер
- Полезные определения:
  - Путь
  - Реберно/вершинно простой путь
  - Цикл
  - Степень вершины

Какие бывают графы

- Ориентированные и неориентированные
- Связные и нет
- Ациклические и нет
- Взвешенные и нет
- Деревья/леса
- Полные графы
- Двудольные графы
- Планарные
- Эйлеровы/Гамильтоновы

Как хранить графы

- Матрица смежности
- Список ребер
- Списки смежности



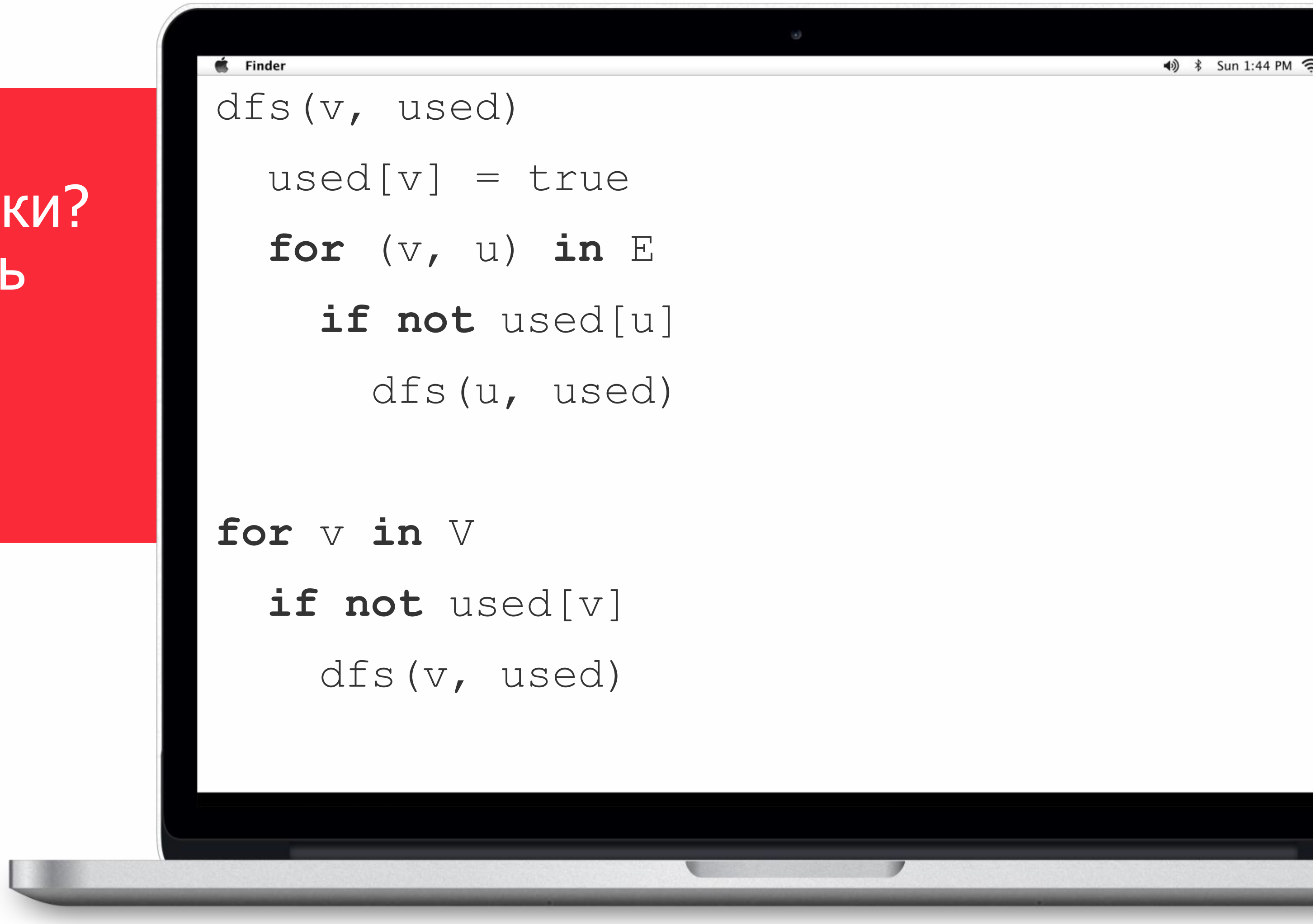
Обход в глубину (DFS)

Обход в глубину

- Рекурсивный алгоритм
- Для каждой вершины смотрим все смежные
- Если находим непосещенную, запускаемся из нее
- В процессе будем получать ребра трех типов:
    - Ребро дерева
    - Прямое
    - Обратное
    - Перекрестное

Обход в глубину

- Что получаем из коробки?
- А как тогда обойти весь граф?
- Сколько работает?
- O(|V| + |E|)

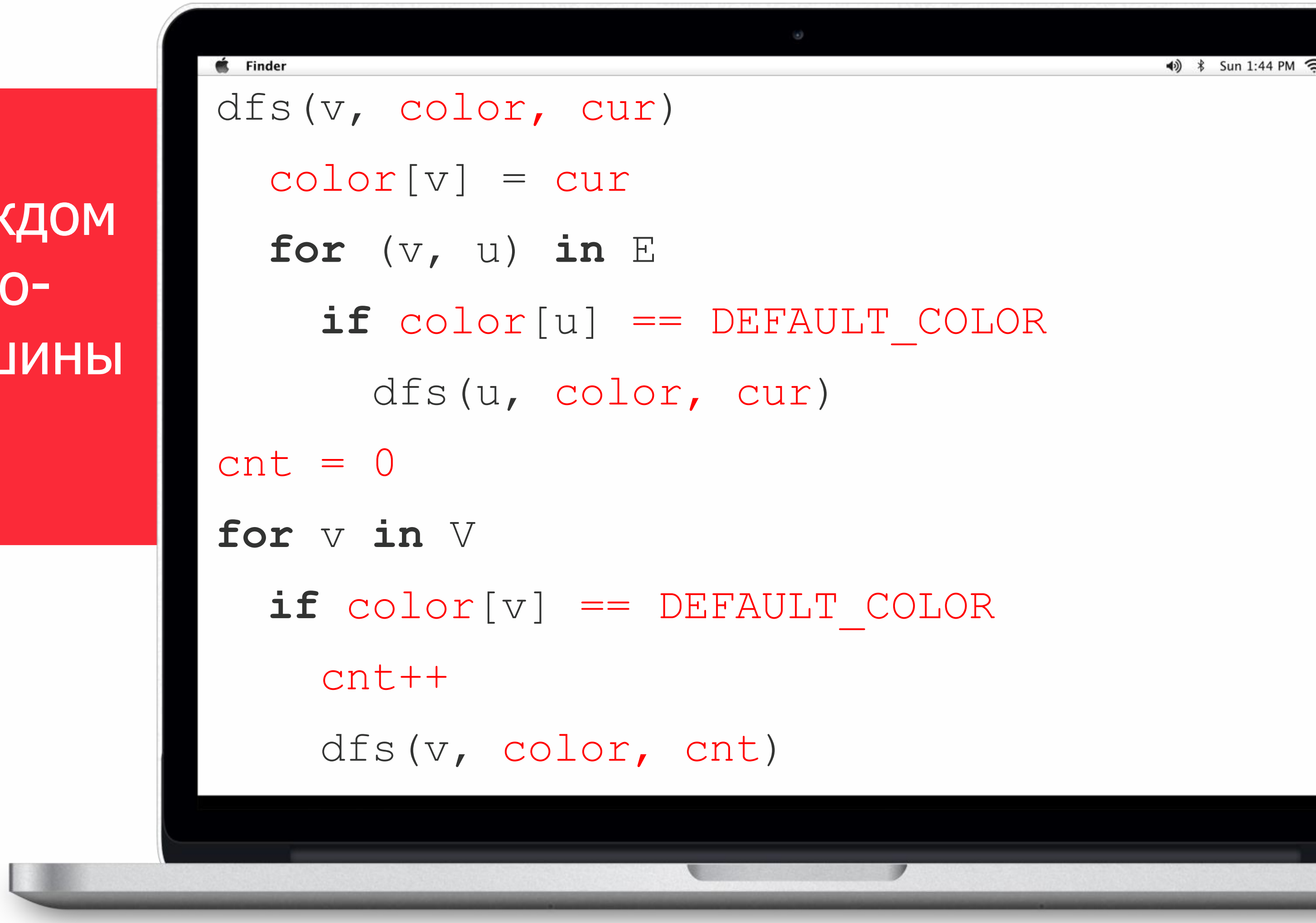


Базовые применения обхода в глубину

- Поиск компонент связности
- Поиск цикла (проверка ацикличности)
- Топологическая сортировка
- Проверка двудольности
- …

Поиск компонент связности

- Давайте будем при каждом рекурсивном запуске по-разному отмечать вершины



Поиск цикла

- Рассмотрим случай ориентированного графа (он сложнее)
- Лемма о белых-серых-черных вершинах
  - Белая – не посещенная
  - Серая – посещенная где-то выше в рекурсии
  - Черная – посещенная, из которой вышли
  - Цикл есть пришли из серой вершины в серую
  - Почему из серой в черную не цикл?
- Что делаем в неориентированном?

Поиск цикла

- Три цвета
- Если цвет u серый, нашли цикл
- Как восстановить цикл?

```
dfs(v, color)
  color[v] = 1
  for (v, u) in E
    if color[u] == 0
      dfs(u, color)
    if color[u] == 1
      Нашли цикл
  color[v] = 2
for v in V
  if color[v] == 0
    dfs(v, color)
```

Топологическая сортировка

- Порядок вершин, в котором нет ребер «справа-налево»
- Если есть цикл, нет топологической сортировки
- Два способа
    - Времена выхода
        - Понятно, что происходит, сложнее писать
    - Второй
        - Непонятно, что происходит, очень просто писать

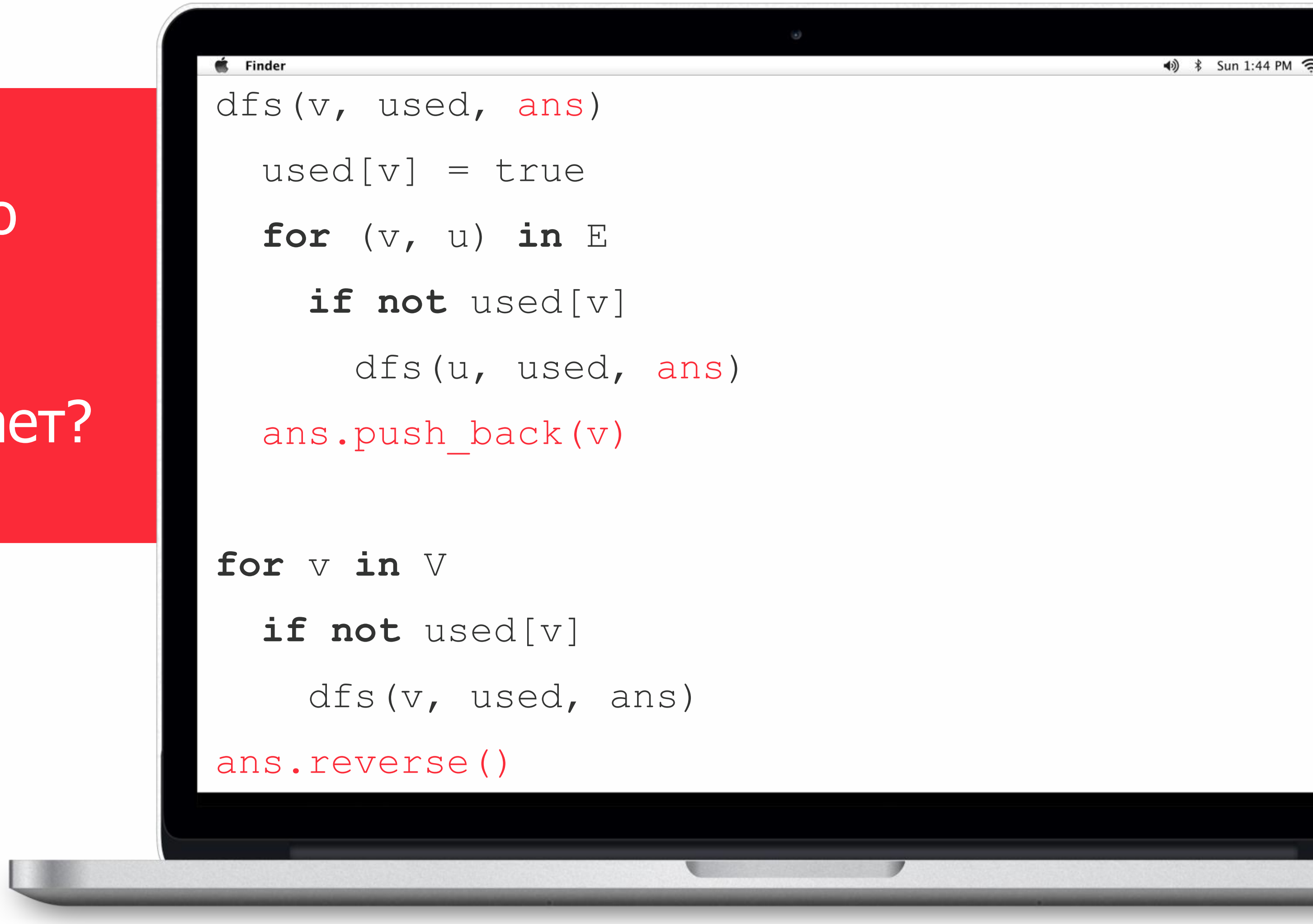
Топологическая сортировка

- Пусть гарантируют, что граф ациклический
- Способ через подсчет времен выхода
- Почему это работает?

```
dfs(v, used, tout)
  used[v] = true
  for (v, u) in E
    if not used[v]
      dfs(u, used, tout)
  tout[v] = T++
T = 0
for v in V
  if not used[v]
    dfs(v, used, tout)
V.sort(by_tout_decrease)
```

Топологическая сортировка

- Пусть гарантируют, что граф ациклический
- Способ «второй»
- А это то почему работает?



Проверка графа на двудольность

- Опять цвета
- Если знаем в какой доле v, однозначно понимаем в какой доле u
- Запускаем обход в глубину
    - Красим
    - Проверяем корректность

Конденсация графа

Конденсация графа

- В неориентированном графе про связность все понятно
- Что в ориентированном?
  - Слабая связность
  - Сильная связность
- *Компонента сильной связности:* наибольшее по включению подмножество вершин, что для любой пары (u, v) из этого множества из u достижима v и наоборот

Конденсация графа

- *Конденсация графа:* граф, в котором каждая вершина соответствует компоненте сильной связности исходного графа, а ребра есть тогда и только тогда, когда в исходном графе были ребра между вершинами соответствующих компонент
- Конденсация – ациклический граф
- Выделить компоненты сильной связности «почти равносильно» нахождению конденсации

Конденсация графа

- Как искать компоненты сильной связности?
- Алгоритм:
  - Построим обратный граф к G
  - $G' = < V', E' >: V' = V;\ E' = \{(u, v) | (v, u)\ in\ E\}$
  - Запустим на исходном графе код для топологической сортировки (звучит некорректно, но так надо)
  - В получившемся порядке запустим код для поиска компонент связности, но на обратном графе
- Утверждение: Если две вершины взаимнодостижимы, после выполнения алгоритма они будут покрашены в один цвет


ARE YOU STUCK ?
WARNING
LOW BRIDGE
10' 7"
Budget
NO OFFICER, I'M DELIVERING A BRIDGE !
imgflip.com

Мосты

Мосты

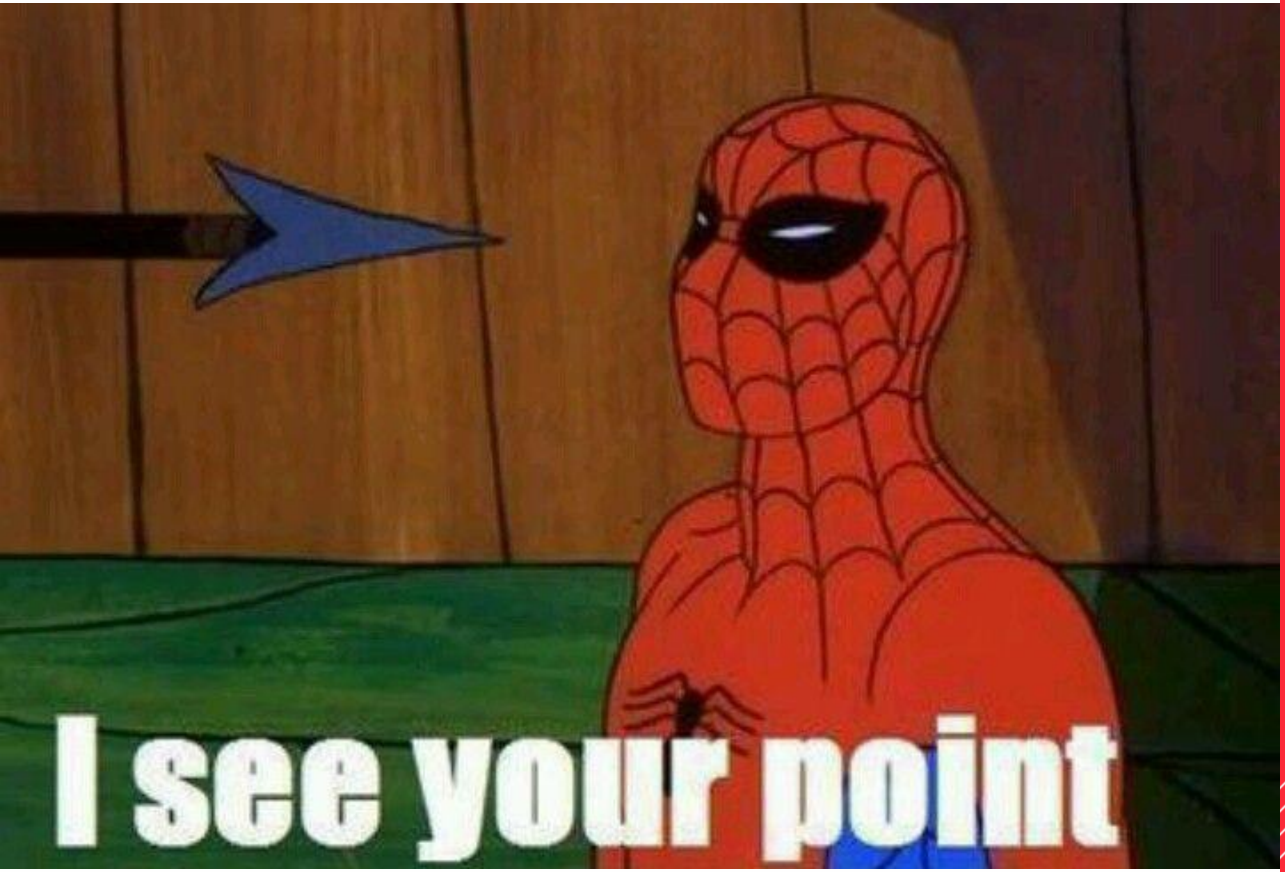
- Мост – ребро, при удалении которого число компонент связности графа увеличивается на одну
- Наивный алгоритм:
  - Попробуем удалить ребро и посчитать число компонент
  - $O(|E|(|V| + |E|)) = O(|E|^2)$
- Давайте придумаем что-нибудь поумнее…

Мосты

- Пусть T – дерево обхода в глубину исходного графа
- Ребро (u, v) – мост тогда и только тогда, когда:
    - (u, v) есть в T
    - Из вершины v или любого ее потомка нет обратного ребра в u или предка u

Мосты

- Введем величину up[v]
- «Как высоко мы можем подняться из вершины v»
- Для оценки высоты посчитаем для каждой вершины время входа в нее tin[v]
- Тогда для up[v] есть рекуррентная формула:
- $up[v] = min\begin{cases} tin[v] \\ tin[p], \text{если есть обратное ребро } (v, p) \\ up[u], \text{если есть ребро дерева } (v, u) \end{cases}$
- Тогда ребро (u, v) мост $\leftrightarrow up[v] > tin[u]$



Точки сочленения

Точки сочленения

- Точка сочленения – вершина, при удалении которой число компонент связности графа увеличивается на одну
- Вершина u точка сочленения:
  - У нее есть ребенок v, из которого не «подняться выше» u
  - Есть ребро (u, v): $up[v] \leq tin[u]$
- Почему этого недостаточно?
  - Корень!

Все!