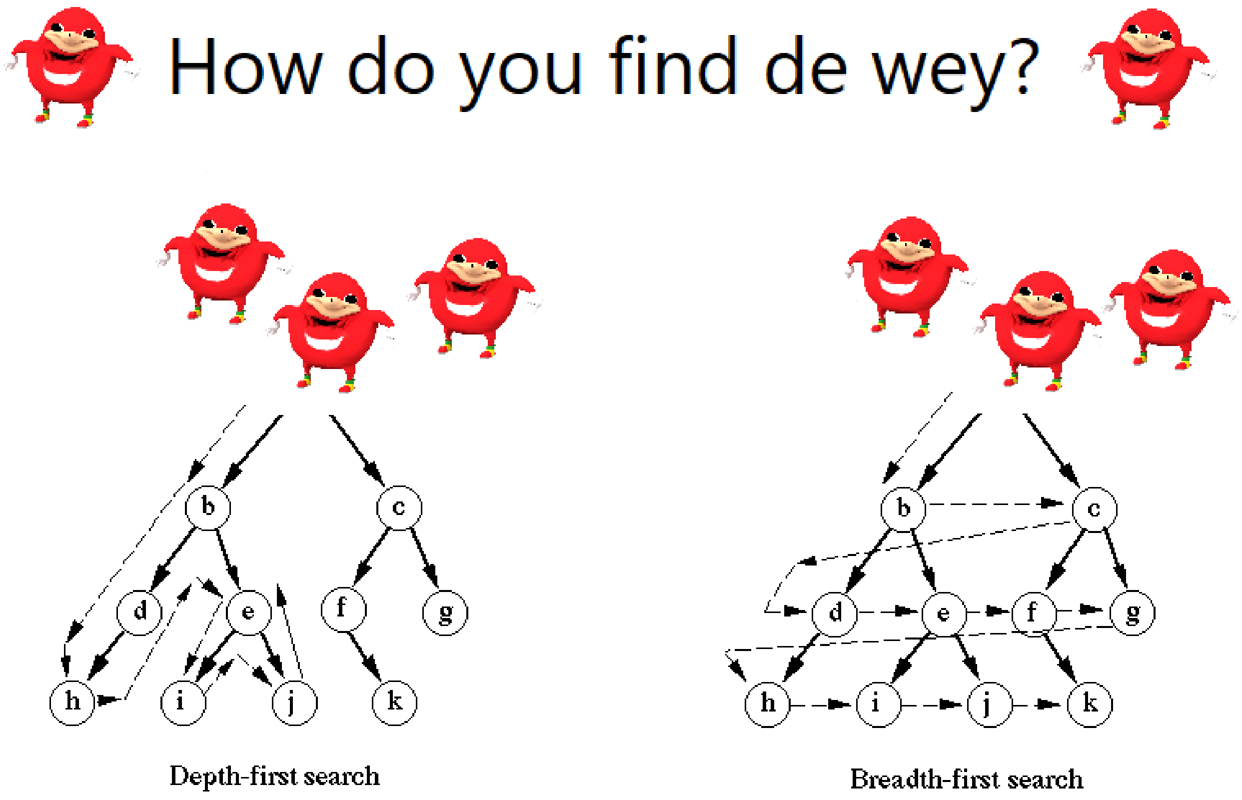
академия больших данных mail.ru group made

Графы – 2. Кратчайшие пути.

Шовкопляс Григорий

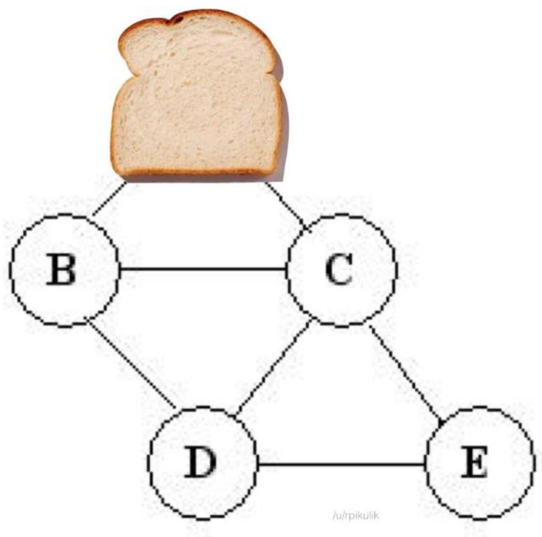
Введение в алгоритмы и структуры данных



Кратчайшие пути в невзвешенном графе

Кратчайшие пути в невзвешенных графах

- Путь: последовательность ребер
- Кратчайший путь: путь, содержащий наименьшее число ребер
- Можно ли искать обходом в глубину?
  - Ну только если в дереве☺
- Есть другой обход!
  - Обход в ширину (Breadth first search)



We are done.

Обход в ширину

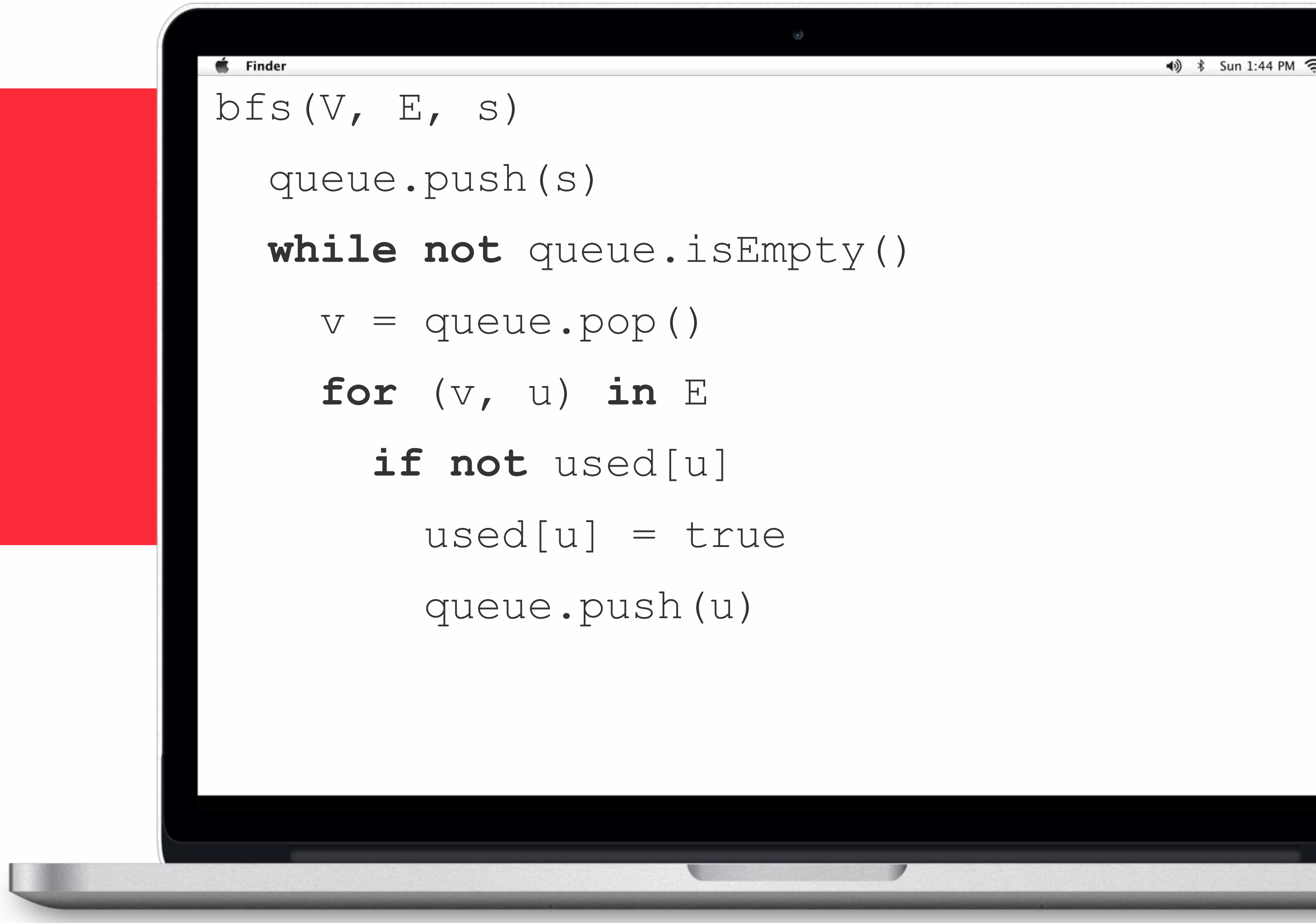
- Раньше: обойдем рекурсивно все, что достижимо из вершины
- Сейчас: обойдем всех соседей, а потом пойдем обходить соседей-соседей
- Как?
    - Очередь достижимых, необработанных вершин
    - Вынимаем из очереди вершину
    - Добавляем всех ее соседей в очередь
    - Если не добавляли раньше

Обход в ширину

- Какие свойства обхода?
- Нерекурсивный
- Обойдем всю компоненту связанности
- Обход в ширину посещает вершины в порядке увеличения расстояния от стартовой

Обход в ширину

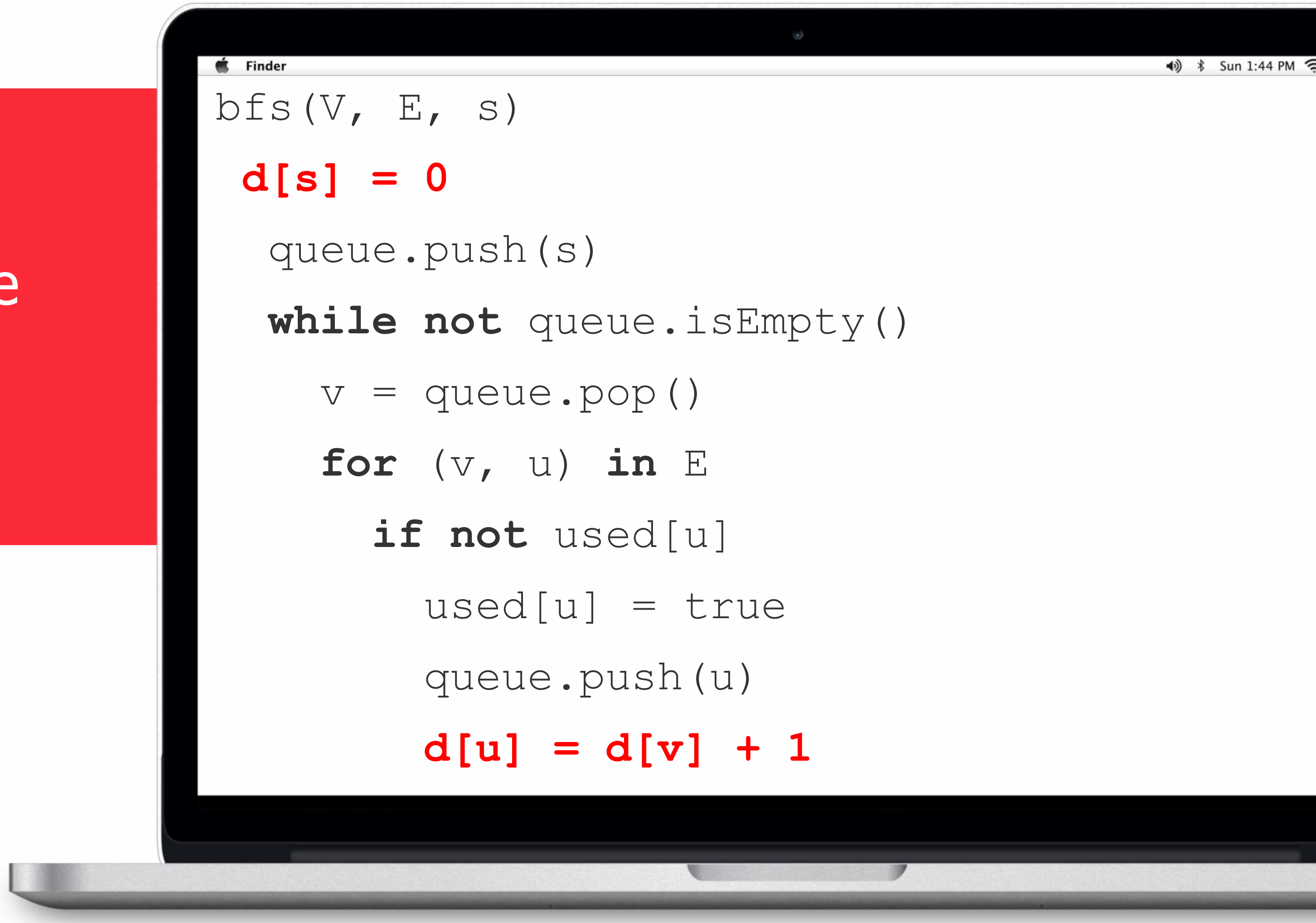
- Сколько работает?
- O(|E|)




```
bfs(V, E, s)
  queue.push(s)
  while not queue.isEmpty()
    v = queue.pop()
    for (v, u) in E
      if not used[u]
        used[u] = true
        queue.push(u)
```

Обход в ширину

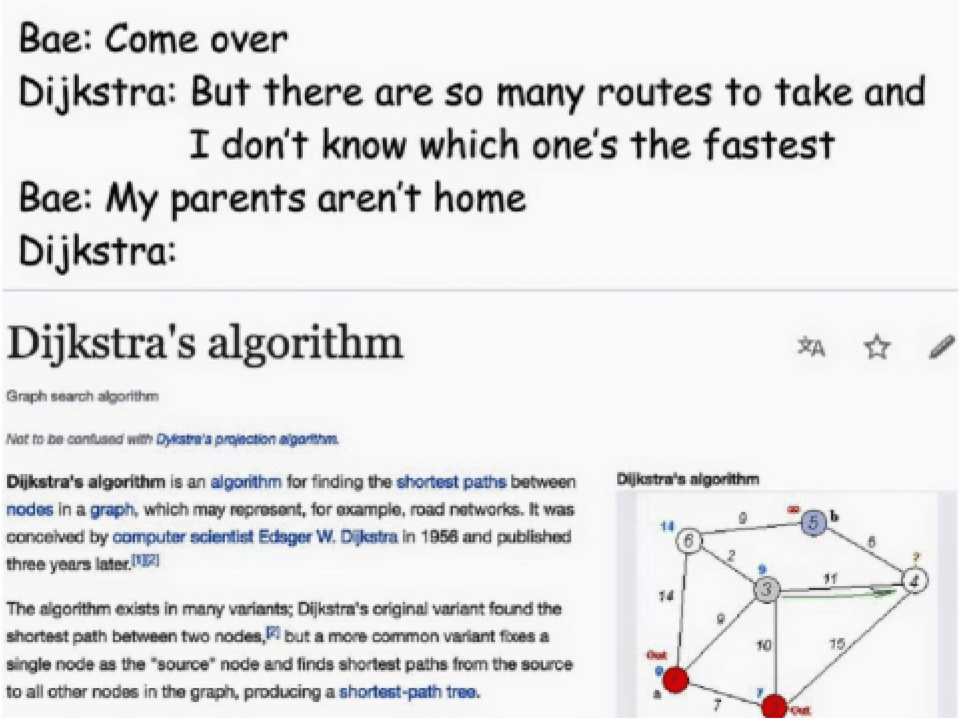
- Можем заодно считать кратчайшее расстояние



Обход в ширину

- Как восстановить путь?
    - Как в обычном динамическом программировании, храним массив предков
- Модификации:
    - 0-1 BFS
    - 1-k BFS

Алгоритм Дейкстры



Dijkstra's algorithm

Алгоритм Дейкстры

- Дан взвешенный граф
- *Все веса неотрицательные!*
- Алгоритм найдет величину кратчайших путей от стартовой вершины для всех остальных
- Кстати, а что такое кратчайший путь во взвешенном графе?
    - Путь, у которого сумма весов ребер минимальна

Алгоритм Дейкстры

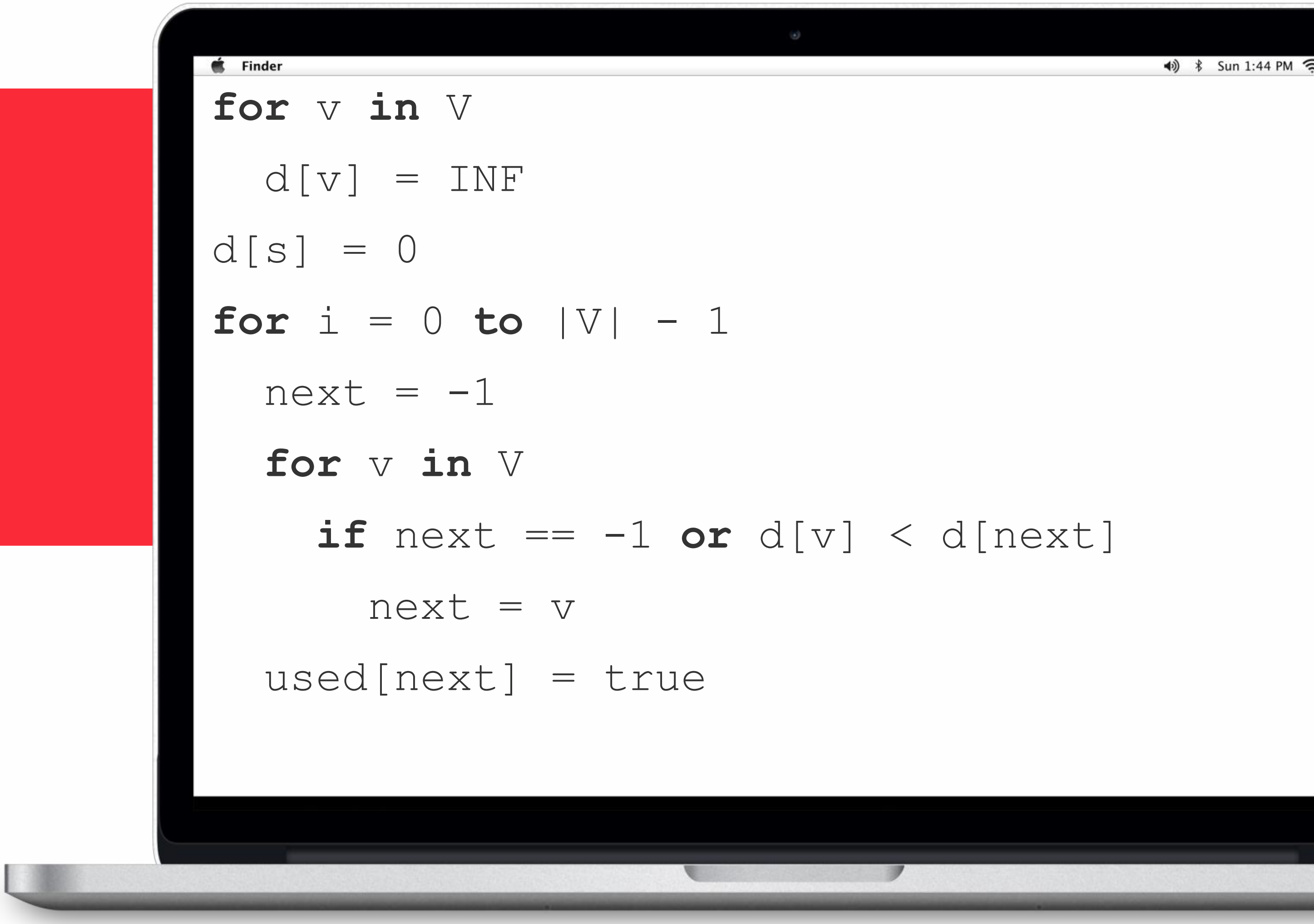
- Инвариант: есть множество вершин U, для которых уже известны кратчайшие расстояния
- Изначально множество U – пустое
- Будем по одной вершине расширять множество U
- Какой переход?
    - Сосед множества U: вершина, которая соединена ребром с вершиной из U, но при этом сама не в U
    - Будем добавлять ближайшего к стартовой вершине соседа множества U
- Будет |V| итераций

Алгоритм Дейкстры

- Докажем корректность
- Почему алгоритм не будет работать, если веса ребер будут отрицательными?
- Окей, идея понятна, а как это все реализовать?
- Будем поддерживать массив кратчайших расстояний до всех вершин
- Множество U может характеризоваться нашим любимым массивом used
- Тогда за O(|V|) сможем найти вершину, которую можно добавить в U

Алгоритм Дейкстры

- Чего-то не хватает!
- Нужно обновить расстояние!

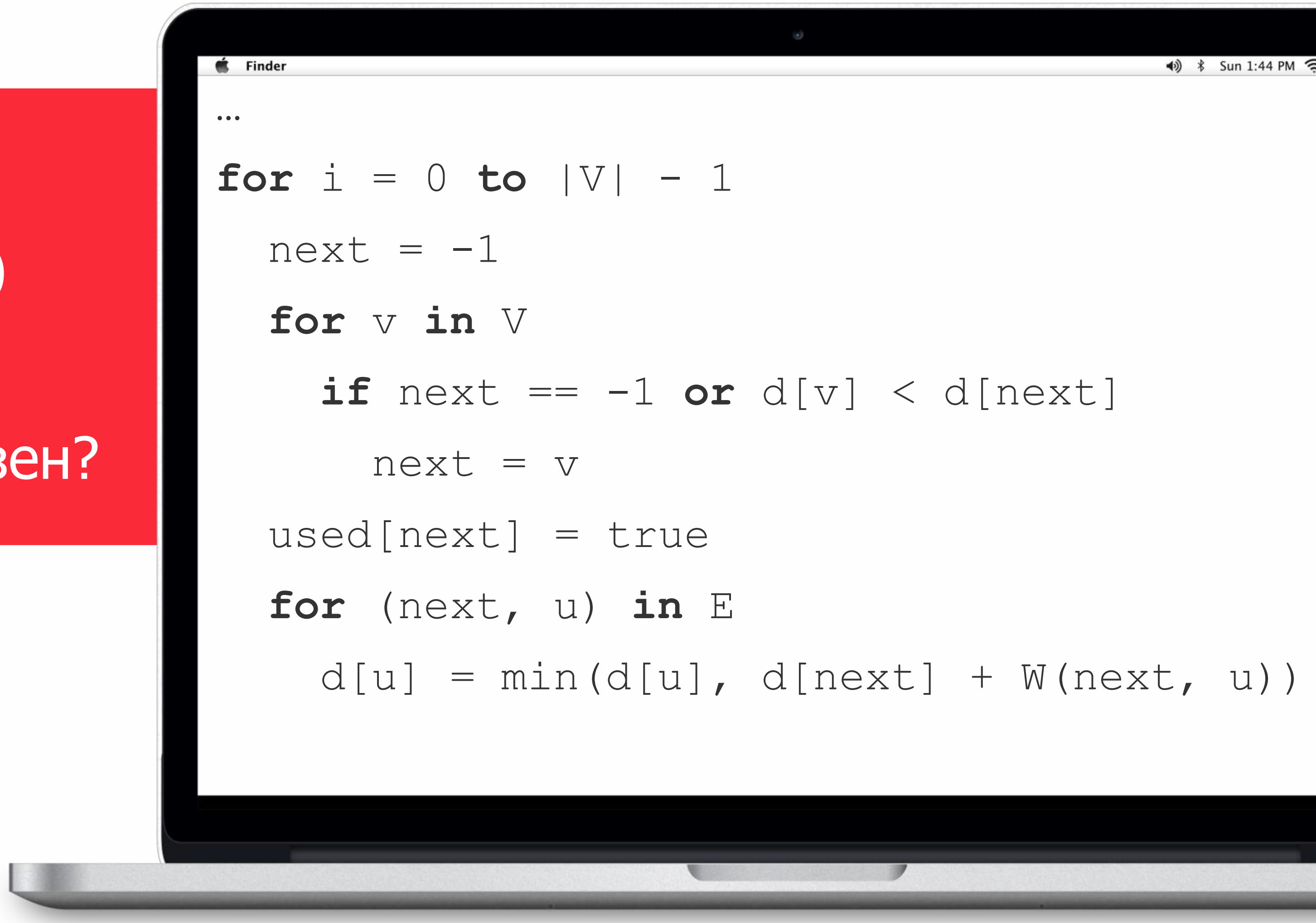


```
for v in V
  d[v] = INF
d[s] = 0
for i = 0 to |V| - 1
  next = -1
  for v in V
    if next == -1 or d[v] < d[next]
      next = v
  used[next] = true
```

Алгоритм Дейкстры

- Сколько работает?
- O($|V^2|$ + |E|) = O($|V^2|$)

- А что если граф несвязен?



Алгоритм Дейкстры

- Добавим условие

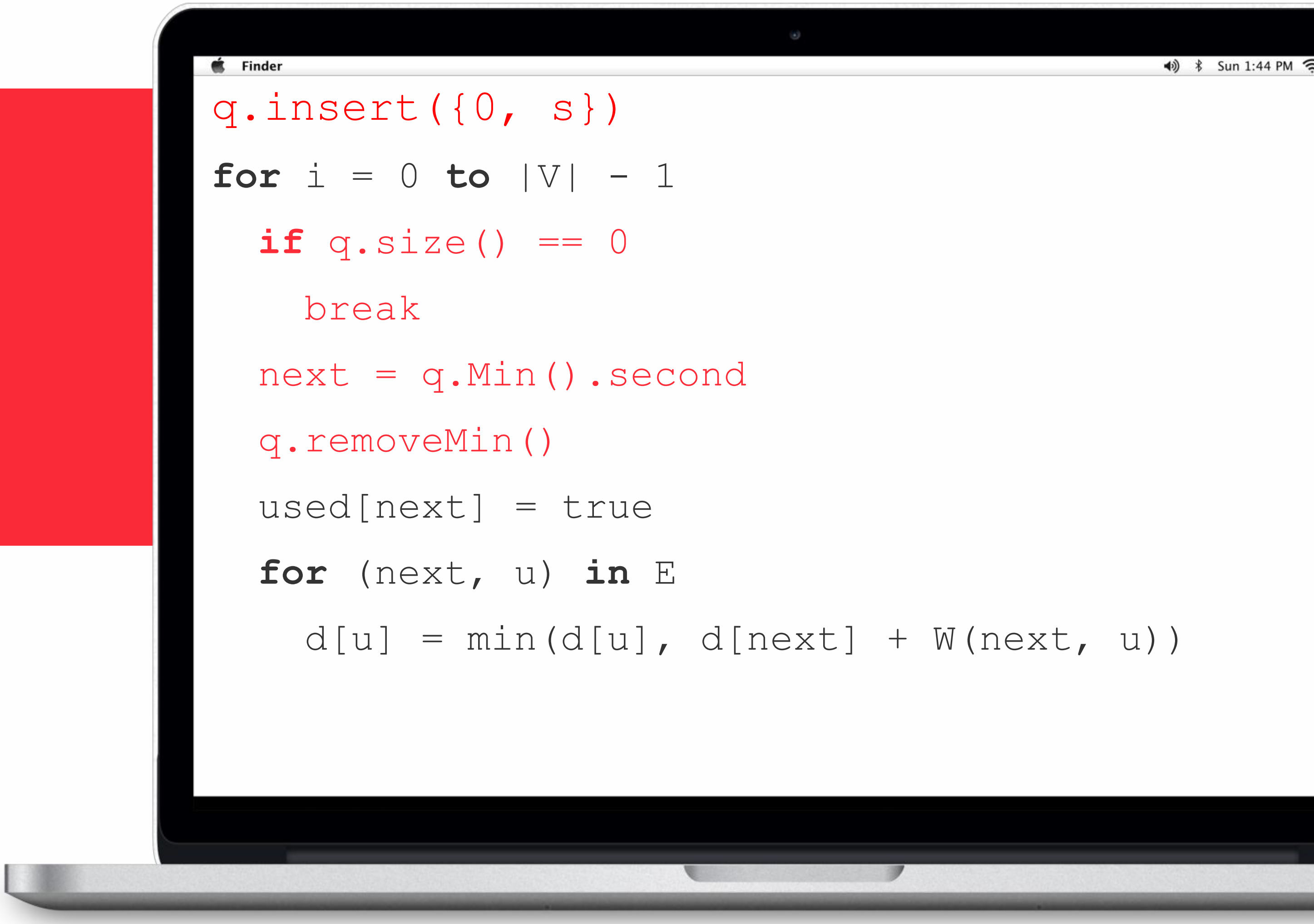
```
...
for i = 0 to |V| - 1
  next = -1
  for v in V
    if next == -1 or d[v] < d[next]
      next = v
  if d[next] == INF
    break
  used[next] = true
  for (next, u) in E
    d[u] = min(d[u], d[next] + W(next, u))
```

Алгоритм Дейкстры

- Как восстановить путь?
  - Как обычно
- Можно ли ускорить?
  - Вроде умеем искать минимум быстрее, чем за линейное время
  - Приоритетная очередь (куча, set…)
  - Поддерживаем множество пар {«за сколько», «куда»}

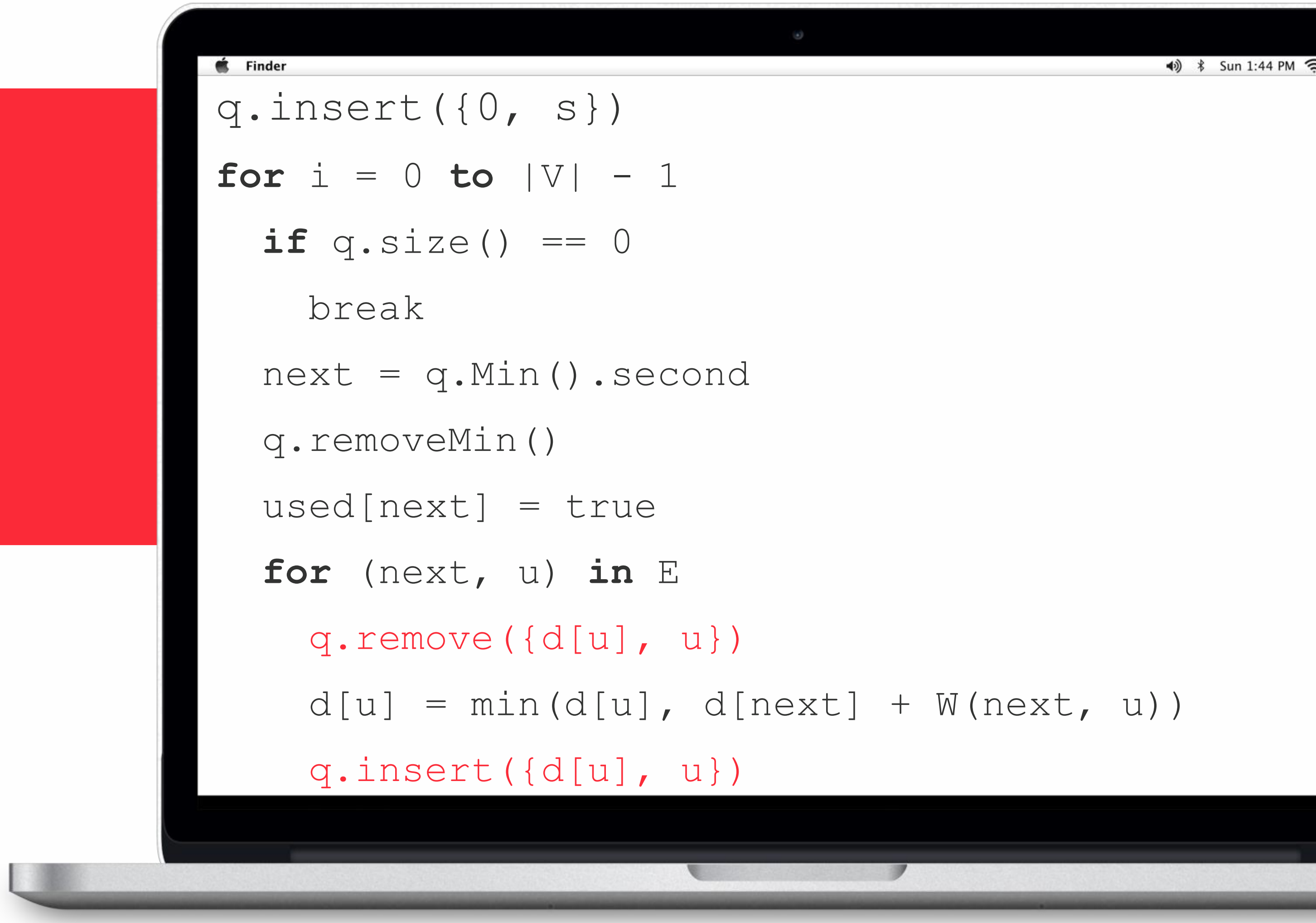
Алгоритм Дейкстры

- Добавим условие
- Ого, это что теперь O(VlogV+E)?
- Не совсем



Алгоритм Дейкстры

- Нужно обновлять расстояния в сете
- Будет работать за O(VlogV+ElogV)



Алгоритм Дейкстры

- Правда ли, что первая вариация алгоритма совсем не нужна, если есть вторая?
- Не совсем
- Есть полные графы!

- Кстати, если изучить такую структуру данных, как Фибоначчиева куча, можно заставить алгоритм работать за O(VlogV + E)

Алгоритм Форда-Беллмана

Алгоритм Форда-Беллмана

- Решим ту же самую задачу, но если бывают ребра с отрицательным весом
- Настало время расчехлять динамическое программирование!
- Количество путей длины k в графе, можно найти с помощью ДП
- $dp[v][k]$ – количество путей длины k в вершину v
- $dp[v][k] = \sum_{(u,v) \in E} dp[u][k-1]$
- Аналогично можем решить и задачу кратчайшего пути
- $d[v][k] = \min_{(u,v) \in E}(d[u][k\ -1] + W(u,v))$
- $d[s][0] = 0, d[i][0] = \infty$

Алгоритм Форда-Беллмана

- Лемма
- Если существует кратчайший путь от s до v, он равен $\min_{k=0..|V|-1} dp[v][k]$
- Что такое «существует кратчайший путь»?

Алгоритм Форда-Беллмана

- Очевидно, работает за O(|V||E|)
- Памяти столько же, можно улучшить?

```
for v in V
  d[v][0] = INF
d[s][0] = 0
for k = 0 to |V| - 1
  for (u, v) in E
    d[v][k] = min(d[v][k],d[u][k-1]+W(u,v)
```

Алгоритм Форда-Беллмана

- На самом деле можно просто забыть про вторую размерность…
- А как понять, что есть цикл отрицательного веса?

```
for v in V
  d[v] = INF
d[s] = 0
for k = 0 to |V| - 1
  for (u, v) in E
    d[v] = min(d[v], d[u]+ W(u,v))

for (u, v) in E
    if d[v] > d[u] + W(u,v)
        Цикл есть!
```

Алгоритм Форда-Беллмана

- Как найти цикл отрицательного веса?
- if d[v] > d[u] + W(u,v): цикл есть (вроде решили)
- Правда ли, что вершина v лежит на цикле?
- Не факт, но она точно достижима с цикла отрицательного веса!
- Если храним массив p, можем вернуться на цикл
- Для всех вершин v_i цикла верно, что $p[v_i] = v_{i-1}$

Алгоритм Форда-Беллмана

- Восстановление цикла отрицательного веса

```
for (u, v) in E
    if d[v] > d[u] + W(u,v)
        for i = 0 to |V| - 1
            v = p[v]
        cur = v
        while p[cur] != v
            ans.add(cur), cur = p[cur]
        ans.add(v)
        ans.reverse()
        break
```

Алгоритм Флойда

Алгоритм Флойда

- А что если нужно найти попарные расстояния для всех вершин?
- Дейкстра: $O(V^3)$ или $O(VElogV)$, но есть нюанс
- Форд-Белман: $O(V^2E)$
- Есть очень простой алгоритм, который все сделает по красоте и без нюансов за $O(V^3)$

Алгоритм Флойда

- Сразу посмотрим код, чтобы кайфануть эстетически

```
d = w
for k in V
  for u in V
    for v in V
      if d[u][v] > d[u][k] + d[k][v]
        d[u][v] = d[u][k] + d[k][v]
```

Алгоритм Флойда

- Снова ДП
- Начнем еще и с трехмерной
- d[u][v][k]: кратчайший путь между вершинами u и v с промежуточными вершинами от 0 до k-1
- d[u][v][0] = w[u][v]
- Тогда логичный переход:
- d[u][v][k] = min(d[u][v][k-1], d[u][k][k-1] + d[k][v][k-1])
- Почему можем отказаться от третьей размерности и сэкономить память?

Алгоритм Флойда

- Как восстановить путь?

```
d = w
next[u][v] = v
for k in V
  for u in V
    for v in V
      if d[u][v] > d[u][k] + d[k][v]
        d[u][v] = d[u][k] + d[k][v]
        next[u][v] = next[u][k]
```

Алгоритм Флойда

- Как найти цикл отрицательного веса?
- $d[v][v] < \ 0 \Rightarrow$ лежит на цикле отрицательного веса
- Что может пойти не так?
- $d[u][v] = d[u][k] + d[k][v]$
- $d[u][v] \downarrow$
- Можно переполниться снизу
- $d[u][v] = \max(-INF, d[u][k] + d[k][v])$

Все!