

# Графы

25/03/2023

# Определение

$$G = (V, E), \quad E \subseteq V \times V$$

$V$  - вершины.

$E$  - ребра.

# Ориентированность

- Неориентированный граф: если есть  $v \rightarrow u$ , то есть  $u \rightarrow v$ . Обычно ребра представлены как  $v \iff u$ .

Пример неориентированного графа: схема метро

Пример ориентированного графа: package dependency

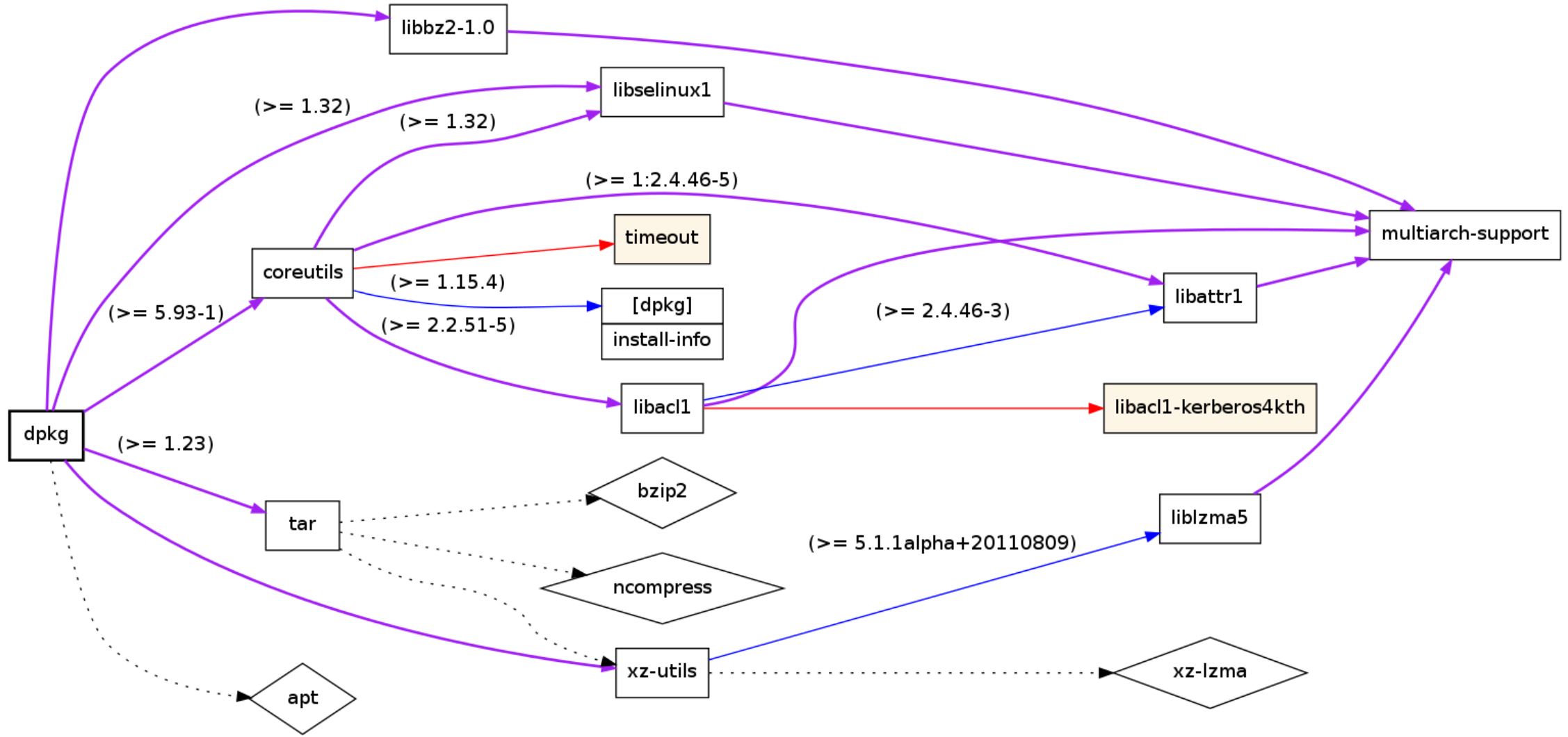


Московский транспорт

Аэропорт  
Шереметьево SVO  
Аэропорт  
Внуково VKO



© Студия Артемия Лебедева



# Взвешенность

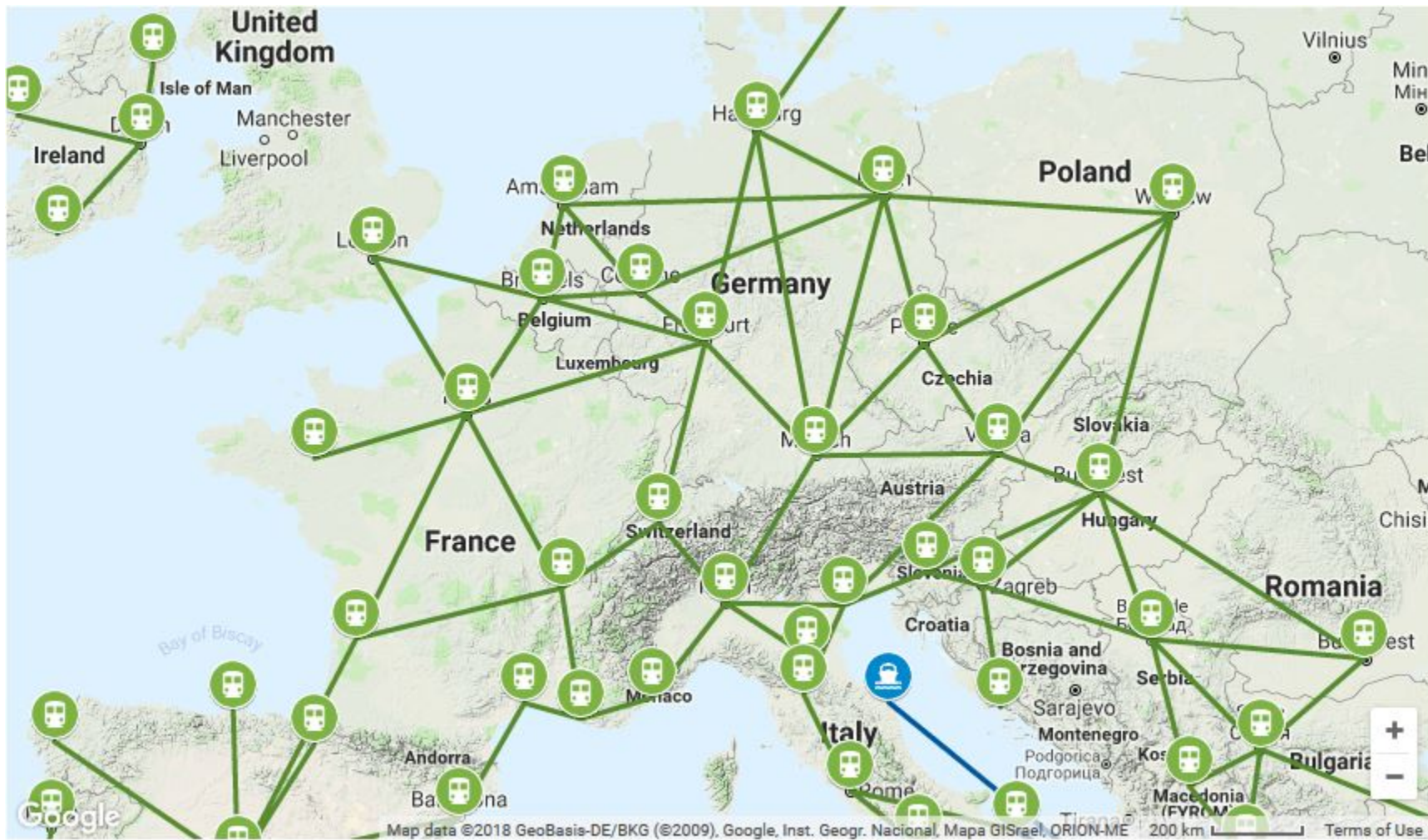
- Взвешенный граф: каждому ребру задан вес  $w$ .

Пример весов: длина дороги

# СВЯЗНОСТЬ

- Связный граф: между двумя вершинами существует путь

Пример несвязного графа: граф железнодорожных путей





# Дерево

Дерево удовлетворяет следующим свойствам:

- $|E| = |V| - 1$
- Связное
- Ациклическое
- Между любыми двумя вершинами существует ровно один простой путь.

# Хранение графа

- Матрица смежности `g = [[0] * n for _ in range(n)]`
- Список смежности

```
g = [[] for _ in range(n)]  
def add_edge(fr, to):  
    g[fr].append(to)
```

# Задача: нахождение пути

$$s \rightarrow t$$

Хотим проверить, что из  $s$  можно добраться в  $t$  переходами по ребрам.

## Подзадача: Обход графа

Хотим пометить все вершины, достижимые из  $s$  как  $used$

Тогда для нахождения пути нужно проверить  $t \in used$

Тогда множество достижимых вершин получается комбинацией объединений.

$$used_v = \{v\} \cup \bigcup_{v \rightarrow u} used_u$$

Проблема - множества зависят друг от друга, нельзя посчитать по определению.

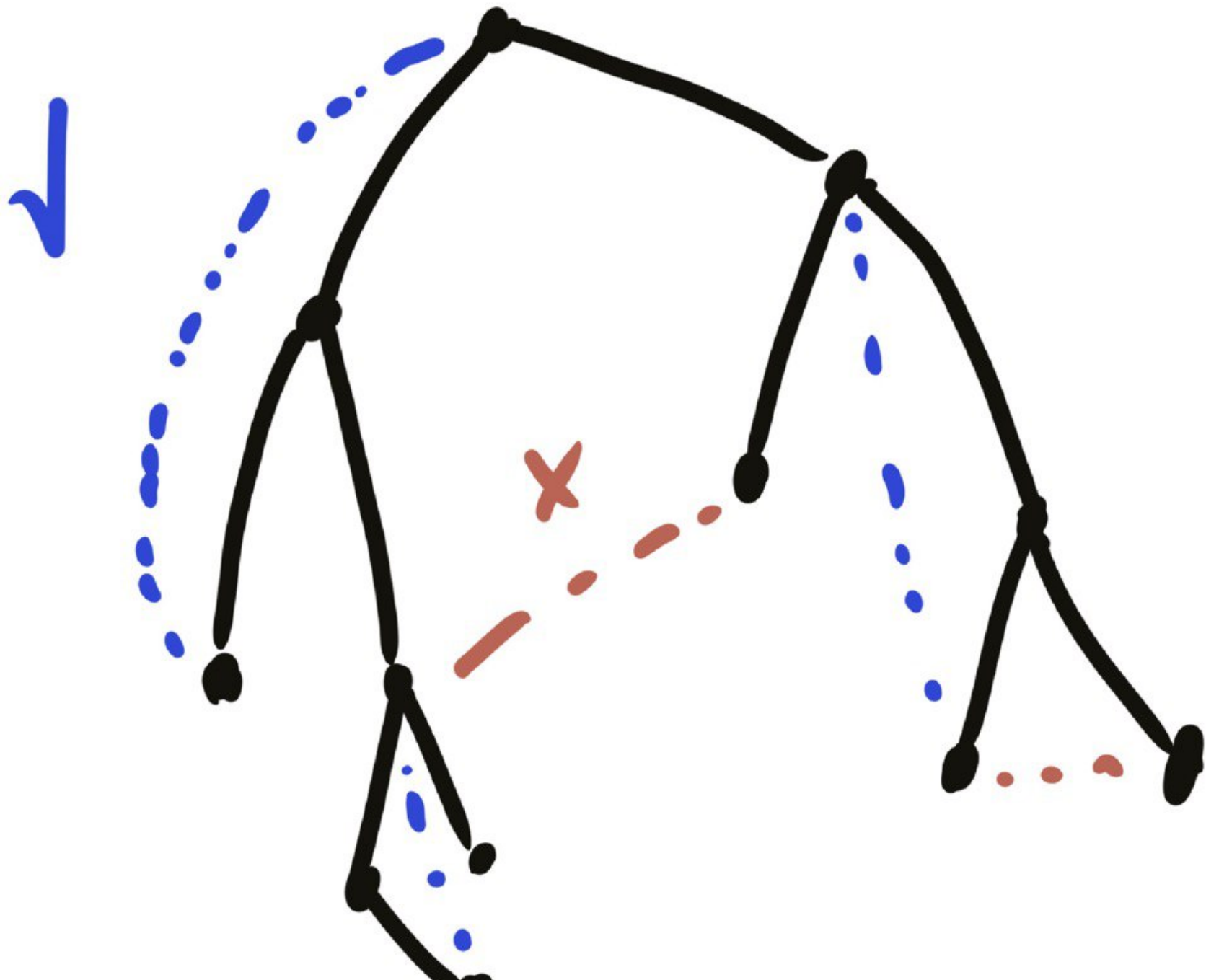
# Обход в глубину

Множества *used* одинаковы вне зависимости от стартовой вершины, поэтому можно просто пометить вершины как доступные

```
def dfs(v):  
    used[v] = True  
    for u in g[v]:  
        if not used[u]:  
            dfs(u)
```

```
dfs(0)
```

**Вопрос:** На каких графах `if` всегда срабатывает?



# Выделение компонент связности

Если запускаться в каждой компоненте по отдельности, можно явно выделить все компоненты.

```
def dfs(v, c):  
    color[v] = c  
    for u in g[v]:  
        if !color[u]:  
            dfs(u, c)  
  
c = 1  
for i in range(n):  
    if !color[i]:  
        dfs(i, c)  
        c += 1
```

# DFS в дереве

DFS-ом удобно считать всякие размеры поддеревьев или глубины с помощью динамик  $sz(v) = 1 + \sum_u sz(u)$ ,  $h(v) = 1 + \max_u h(u)$

```
def dfs(v, p=-1):  
    sz[v] = 1  
    for u in g[v]:  
        if u != p:  
            sz[v] += dfs(u, v)  
    return sz[v]  
  
dfs(0)
```



## Асимптотика

Мы не запускаемся из одной вершины дважды, поэтому всего запусков  $O(|V|)$ .

Внутри каждого запуска мы перебираем  $deg_v$  ребер. Мы знаем, что  $\sum deg_v = 2|E|$ , поэтому общая сложность алгоритма  $O(|V| + |E|)$ .

# Оптимальные расстояния

Хотим находить кратчайшие пути в графе.

$$dist_v = \min_{v \rightarrow u} (dist_u + w(v, u))$$

Опять же, все числа зависят друг от друга.

## Задача: Поиск пути

Хотим найти путь от  $s$  до  $t$ , содержащий минимальное количество ребер.

## Кратчайшие пути: идея

Давайте поддерживать множество оптимальных вершин. Для них мы будем знать финализированное кратчайшее расстояние.

Для остальных вершин мы можем *релаксировать* ответ по формуле.

Самый примитивный алгоритм - пока можно уменьшать какое-либо  $dist_*$ , релаксируем ответ. Когда ничего больше нельзя релаксировать, мы нашли кратчайшие пути.

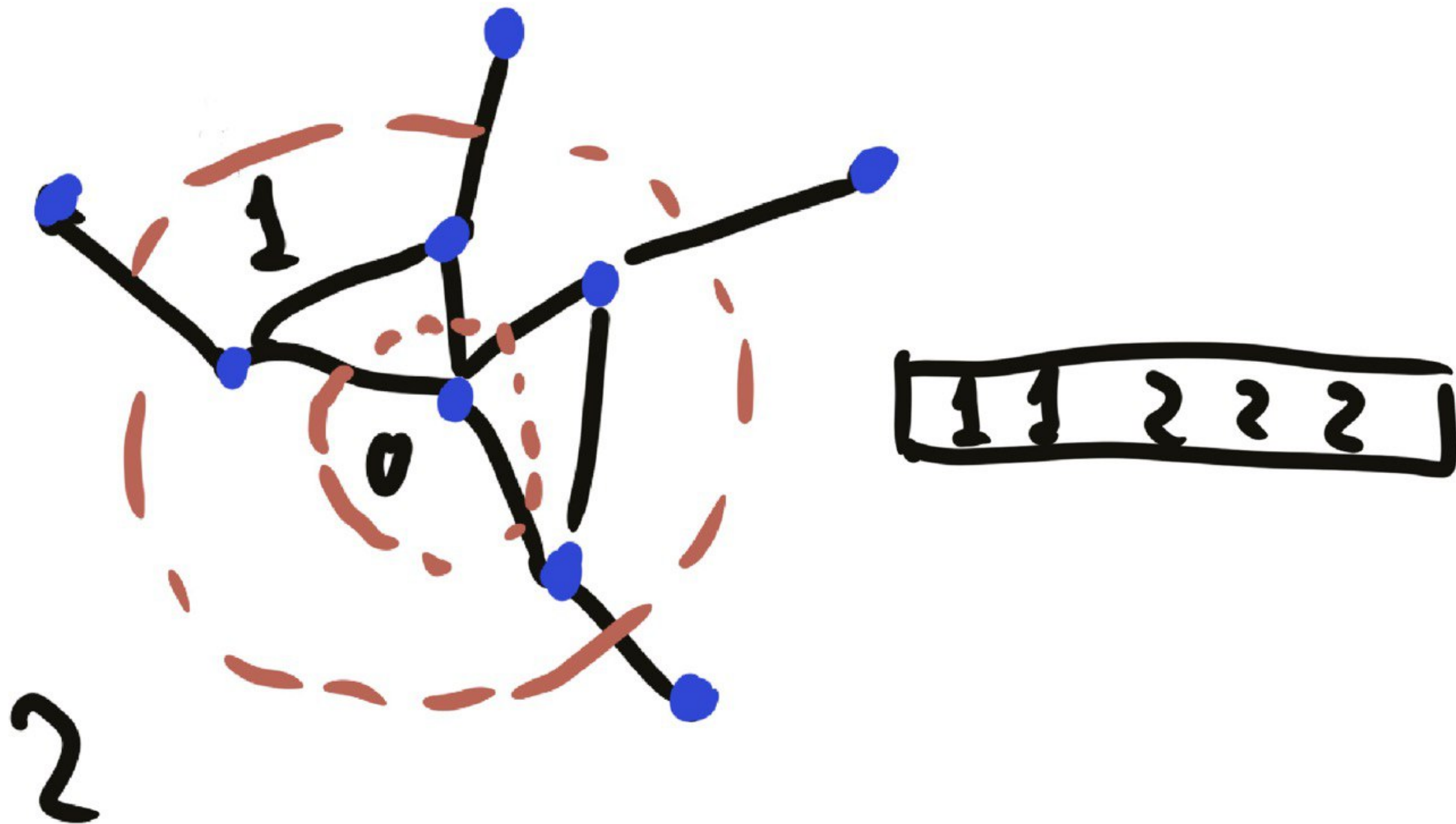
## Обход в ширину

Для множества оптимальных вершин можно поддерживать уже посчитанные расстояния.

Для остальных вершин можно поддерживать промежуточные значения.

Переход вершины из промежуточного значения в финальное будем делать через очередь: вершина кладется в очередь тогда, когда мы знаем ее финальное состояние.

Таким образом, финальное множество постепенно расползается.



# BFS

```
def bfs():  
    dist = [INF] * n  
    dist[s] = 0  
    q = Queue([s])  
    while len(q) > 0:  
        v = q.pop()  
        for u in g[v]:  
            if dist[u] == INF:  
                dist[u] = dist[v] + 1  
                q.push(u)
```

Асимптотика  $O(|V| + |E|)$

# Алгоритм Дейкстры

Для поиска кратчайших расстояний во взвешенных графах существует алгоритм Дейкстры. На него можно смотреть, как на обобщение bfs с помощью очереди с приоритетами (кучи).

Мы храним финальные вершины, и по одной переносим вершины из множества промежуточных в множество финальных.

Асимптотика  $O((|E| + |V|) \log n)$  - берем асимптотику BFS и добавляем работу с кучей.



# Алгоритм Дейкстры: реализация

```
def bfs():
    dist = [INF] * n
    dist[s] = 0
    q = Heap(zip(dist, range(n)))

    while len(q) > 0:
        d, v = q.pop_min()
        for u, w in g[v]:
            if dist[u] > d + w:
                q.erase((dist[u], u))
                dist[u] = d + w
                q.push((dist[u], u))
```

**Вопрос:** сколько случится итераций внешнего цикла? Какая проверка нужна, чтобы их стало  $n$ ?

## Вопрос

Работает ли Дейкстра с отрицательными ребрами?

# Отрицательные циклы

Отрицательные ребра рушат условие  $dist(v, v) = 0$ , потому что по отрицательному циклу можно уменьшать расстояние бесконечно.

# Алгоритм Форда-Беллмана

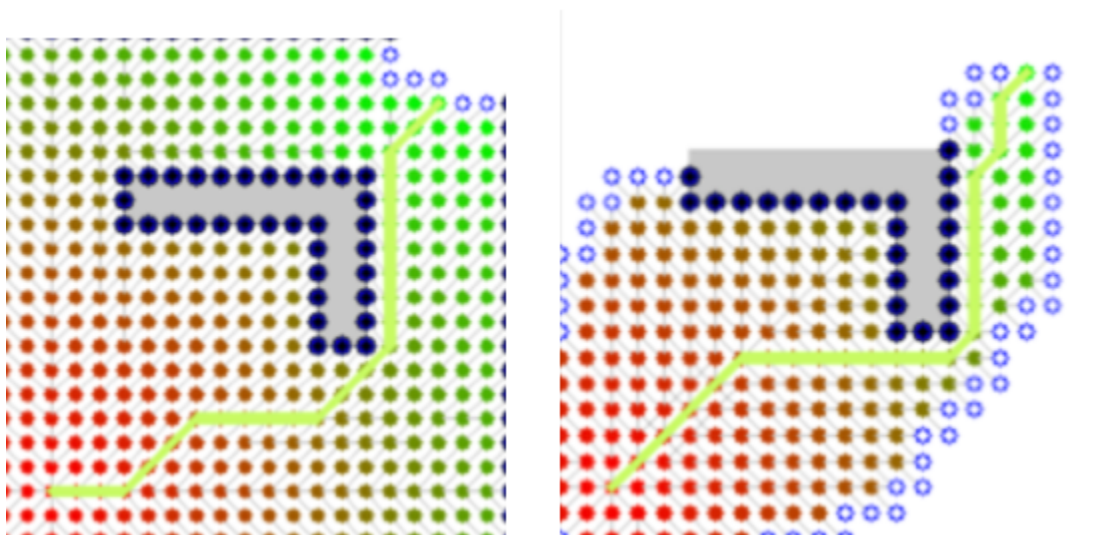
Просто всегда релаксирует ответ. Если на  $n + 1$ -м шаге есть что релаксировать - мы нашли отрицательный цикл.

```
dist = [INF] * n
dist[s] = 0

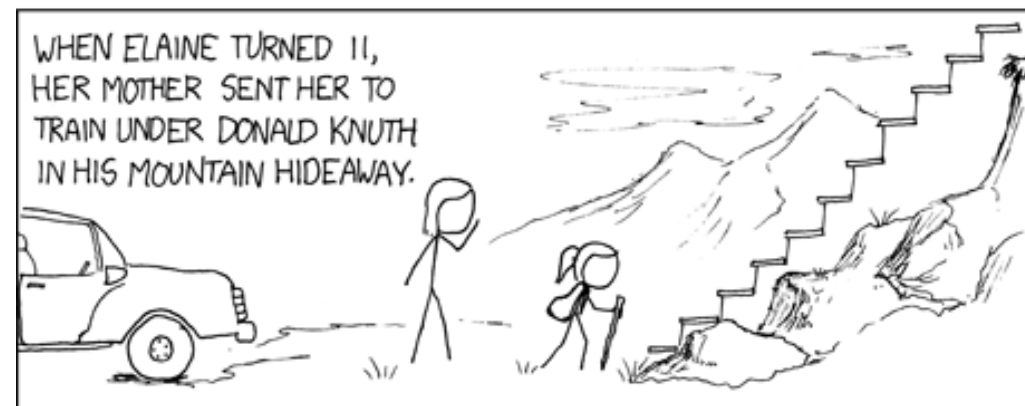
for i in range(n + 1):
    for fr, to, w in edges:
        if dist[to] > dist[fr] + w:
            dist[to] = dist[fr] + w
```

# A\*

Эвристический алгоритм для реальных графов, который вместе с расстоянием использует оценочную функцию (например, евклидово расстояние между вершиной и финишем). Вершины, имеющие меньшую сумму  $dist(s, v) + expected\_dist(v, t)$  оцениваются раньше в очереди с приоритетами.



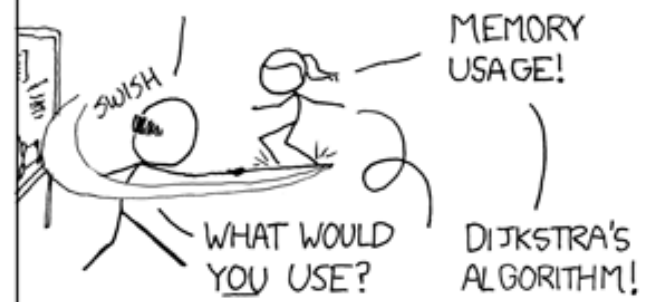
# Mem



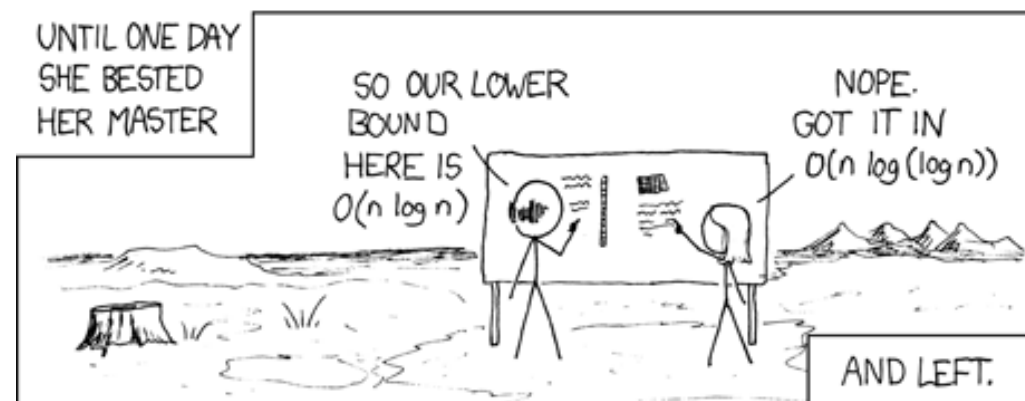
FOR FOUR YEARS SHE  
STUDIED ALGORITHMS.



WHY IS A\* SEARCH WRONG  
IN THIS SITUATION?

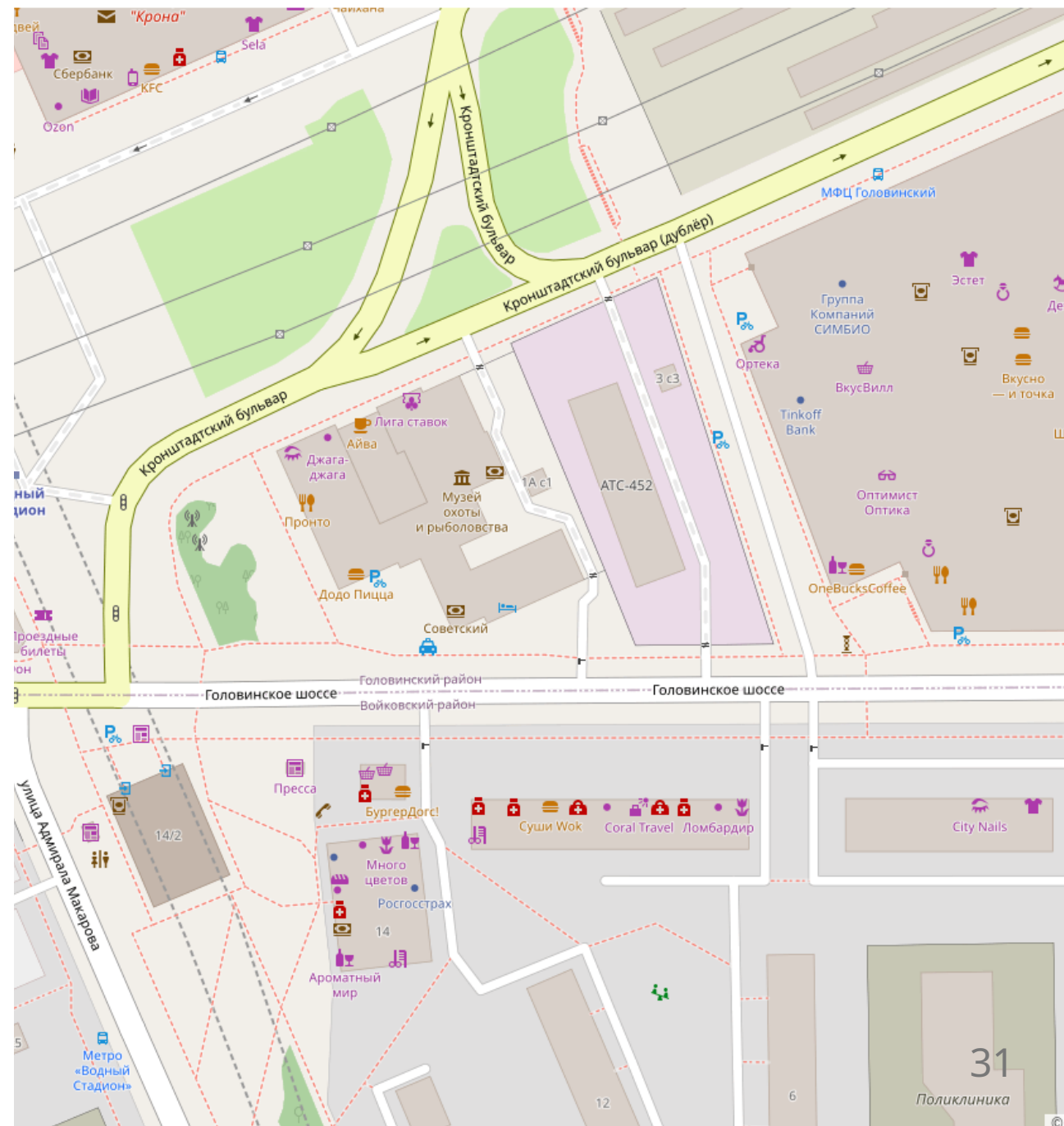


UNTIL ONE DAY  
SHE BESTED  
HER MASTER



# Идея для пет-проекта

Экспортировать карту своего района (например, в OSM) и искать в ней кратчайшие пути от дома до универса, магазина, поликлиники и прочего.



# Для дальнейшего изучения

- Сходите на семинары!
- Дискретная математика
- Двудольные графы, паросочетания
- Задача о максимальном потоке
- Минимальный остов
- Задача коммивояжера