Интеллектуальные информационные системы

Работа с графами

Кафедра управления и интеллектуальных технологий НИУ «МЭИ»

2023 г.

Что такое графы

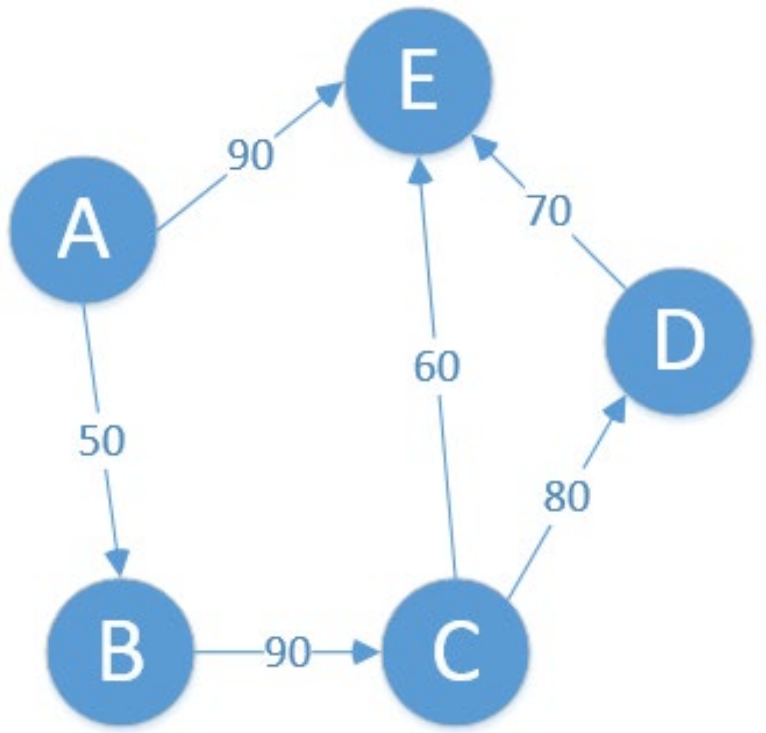
G(V,E)

- V - множество вершин (узлов, nodes)
- E – множество ребер (edges, links)
- V,E могут содержать дополнительную информацию, например:
- W(vi,vj) – вес ребра – стоимость прохода по ребру

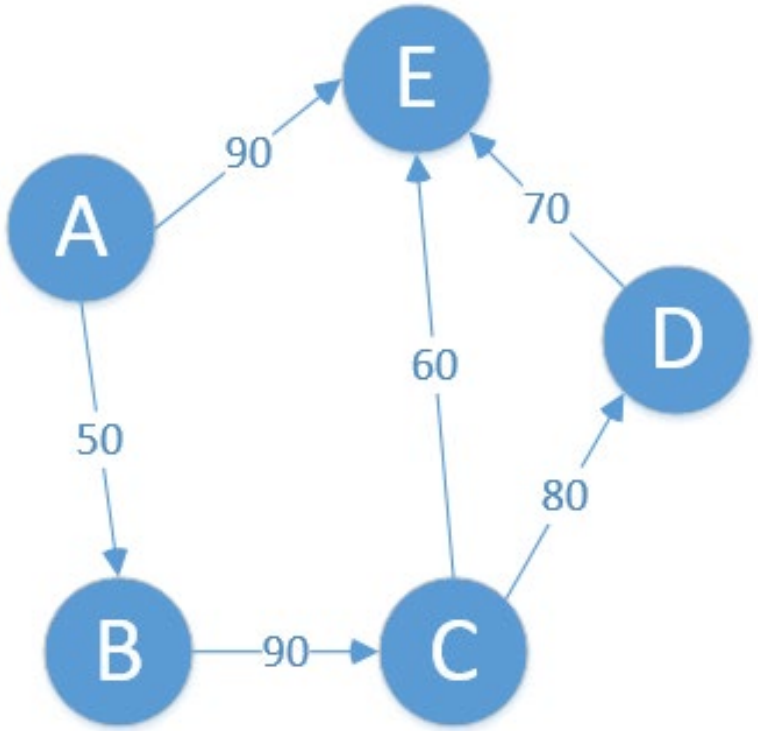
V = {A, B, C, D, E}

E = {[A,B], [B,C], [A,E], [C,E], [C,D], [D,E]}

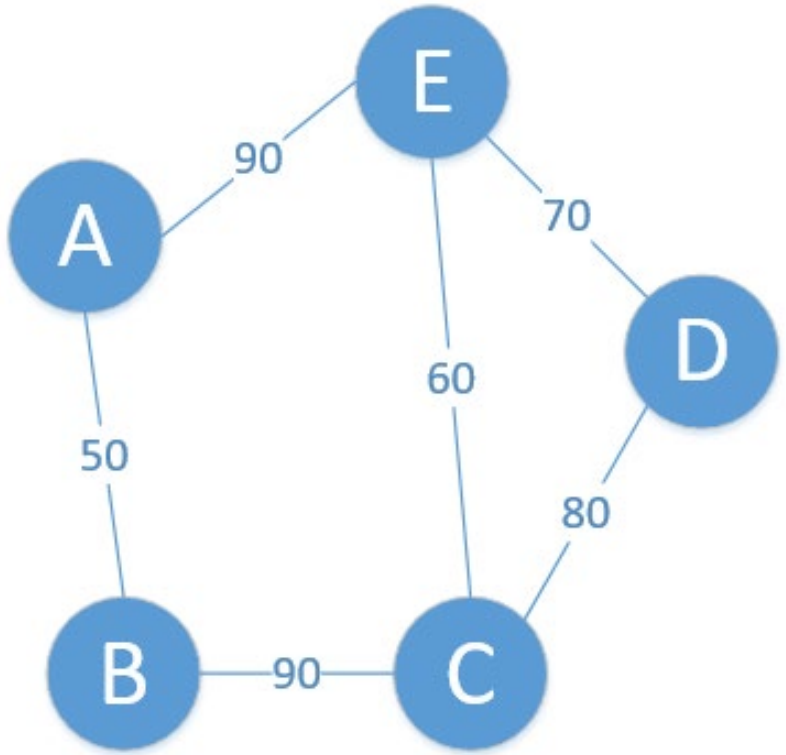
$W_{AB} = 50$, $W_{BC} = 90$, $W_{AE} = 90$, $W_{CE} = 60$, $W_{CD} = 80$, $W_{DE} = 70$



Типы графов

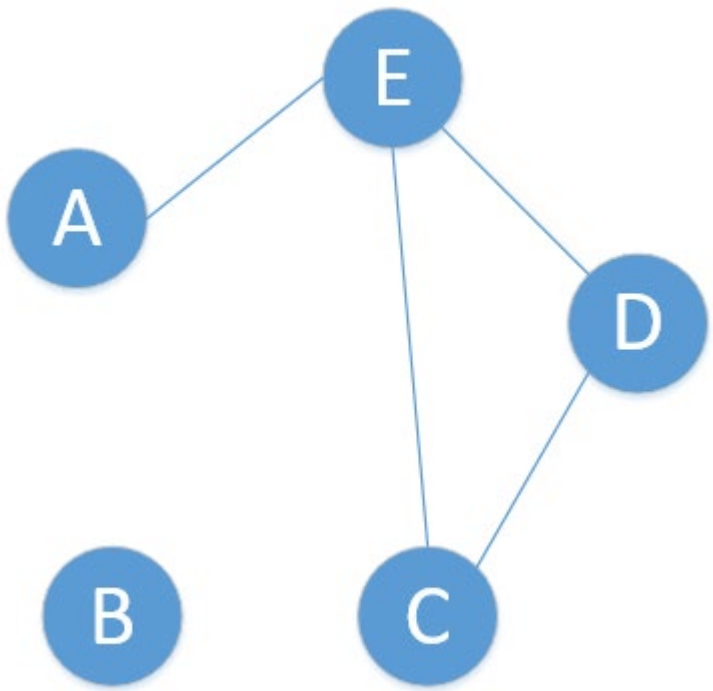


Ориентированный

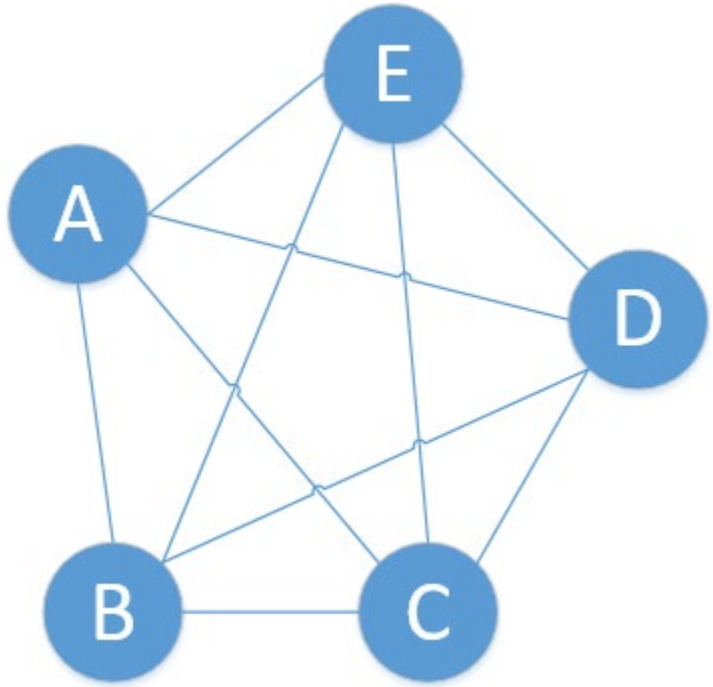


Неориентированный

Типы графов

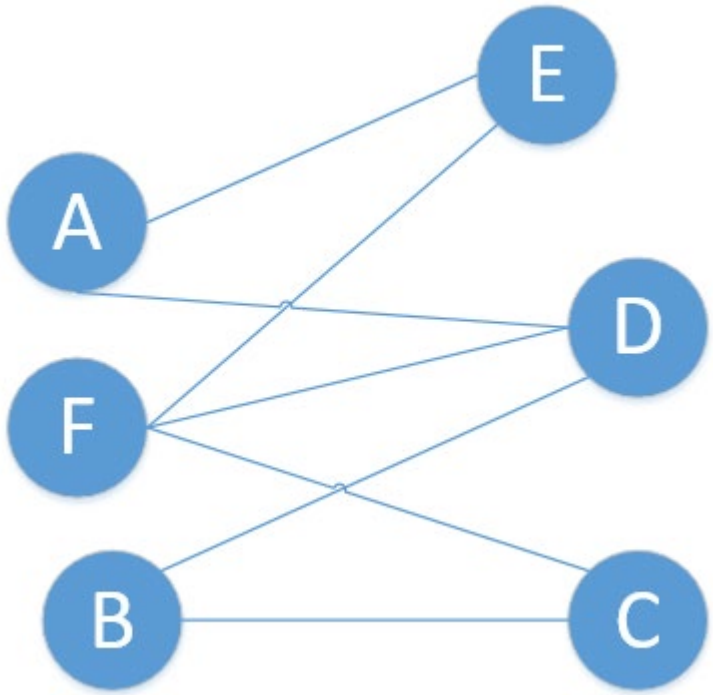


Несвязный



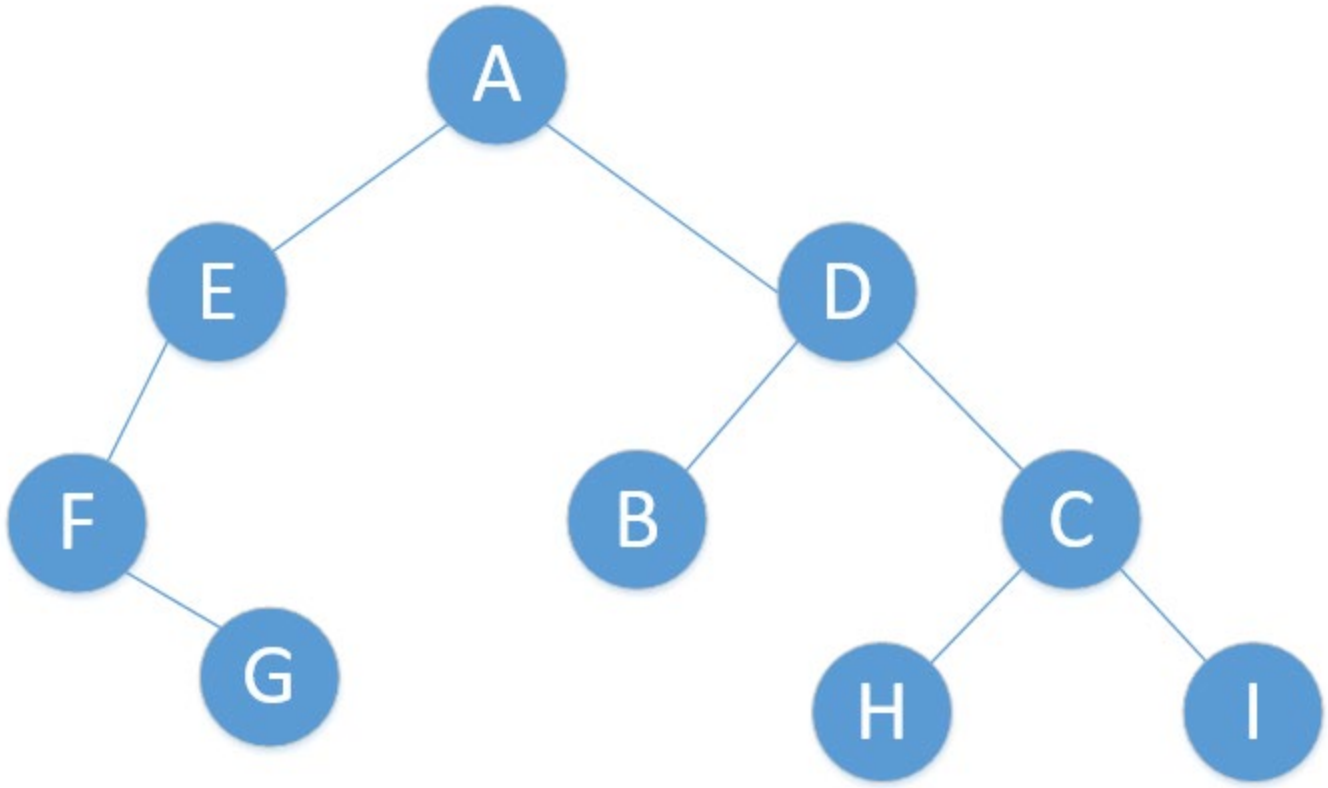
Полный

Типы графов



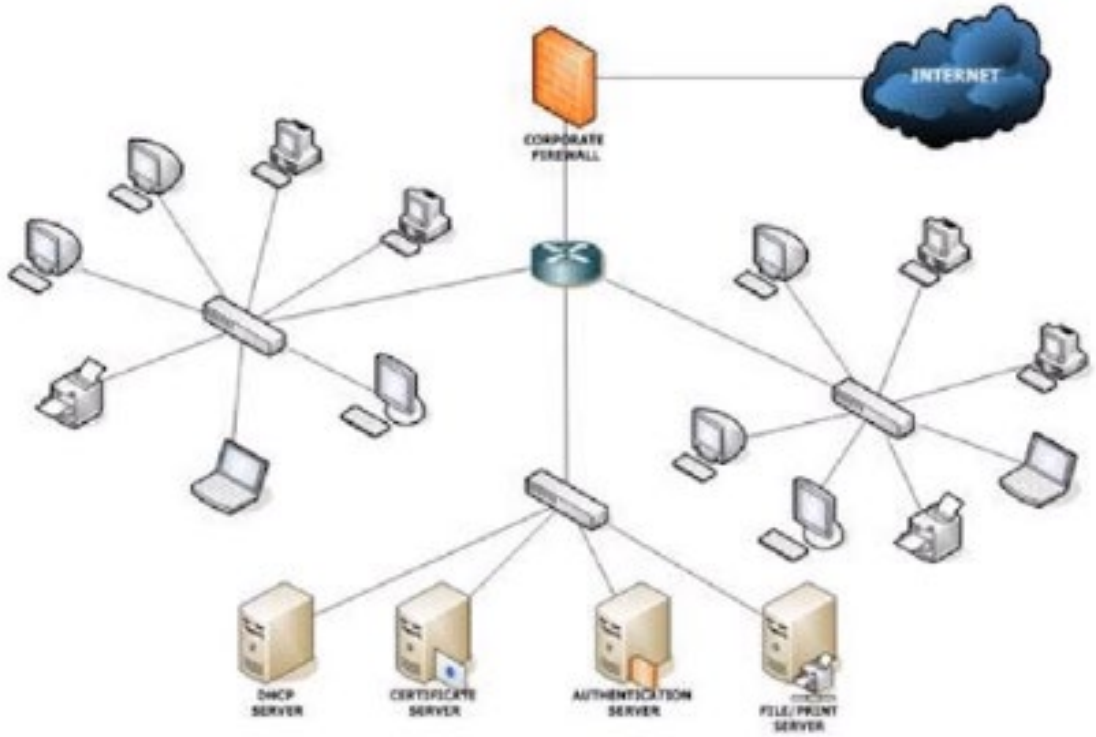
Двудольный

Типы графов



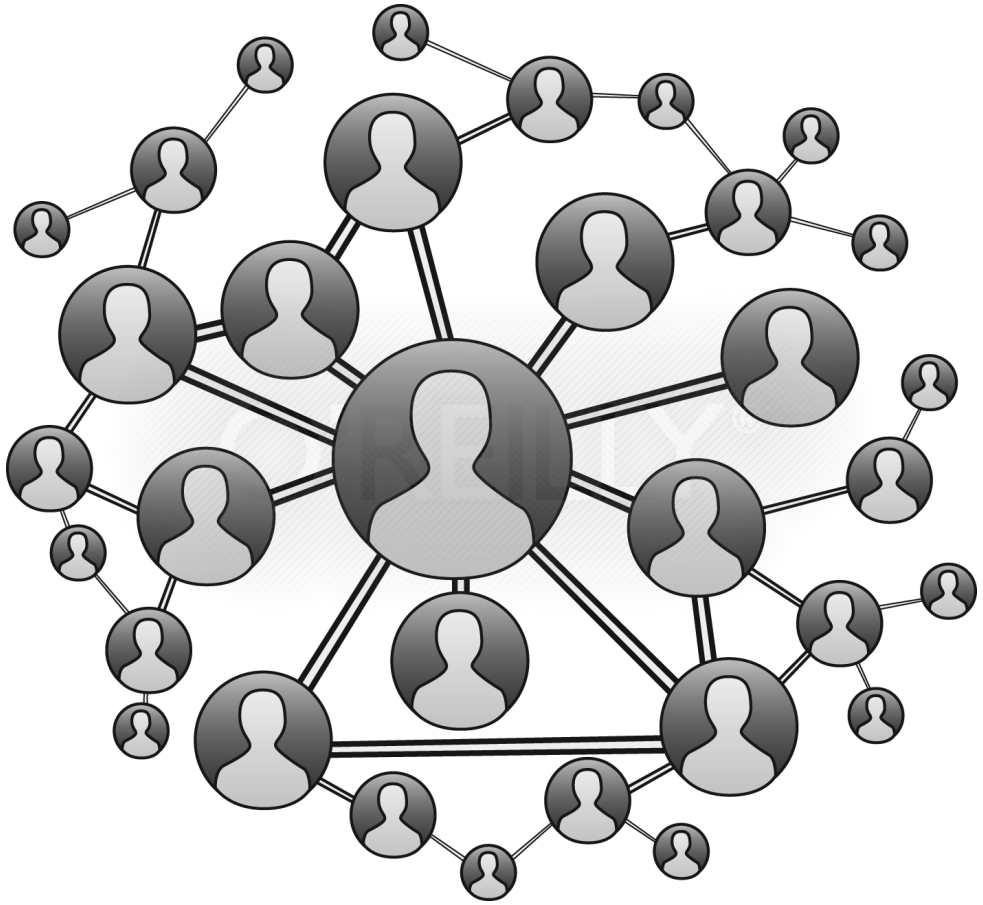
Дерево

Где используются графы



Структура сети

Где используются графы

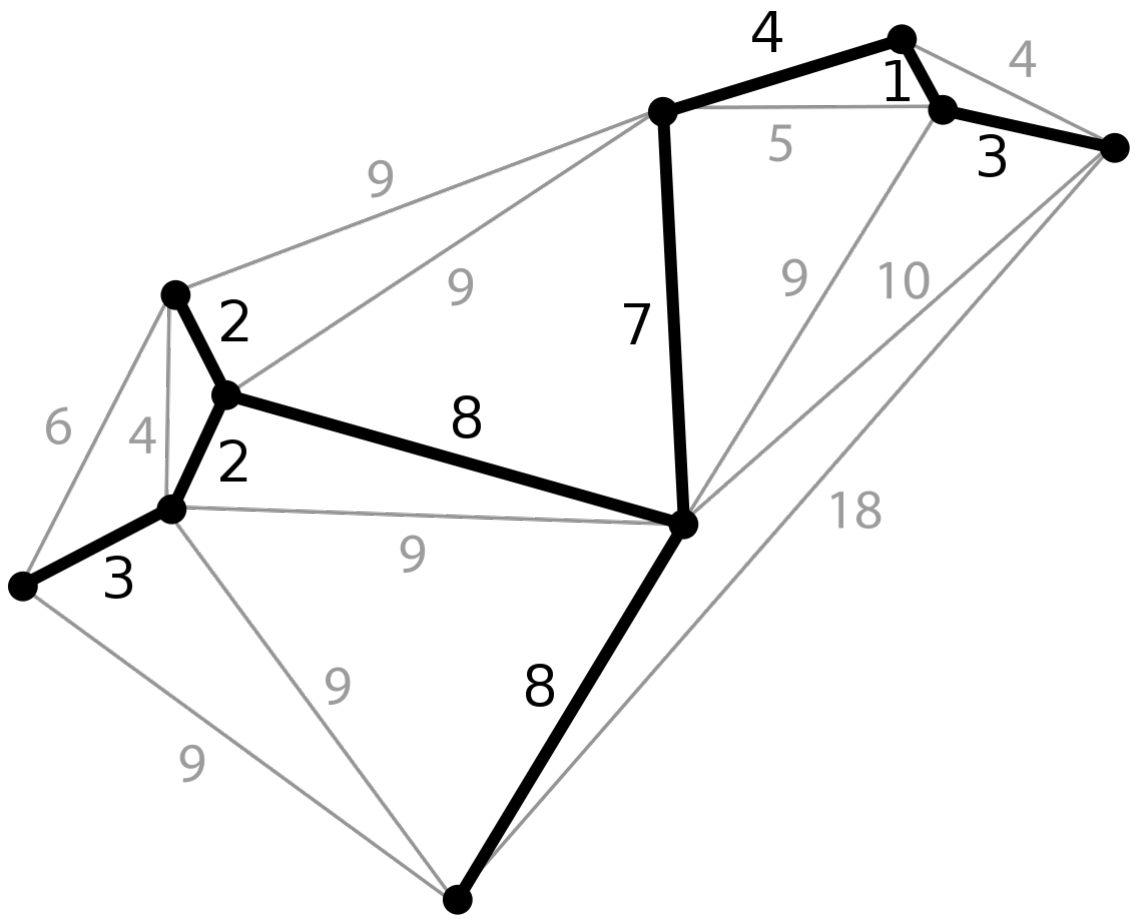


Социальный граф

Где используются графы

Построение маршрута

Где используются графы



Минимальное остовное дерево (minimum spanning tree)

Есть *n* городов, которые необходимо соединить дорогами так, чтобы можно было добраться из любого города в любой другой

Где используются графы

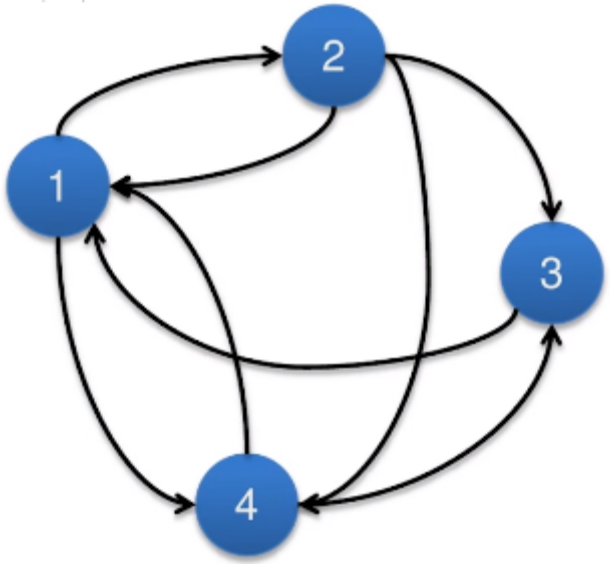
Определение «важности» веб-страницы. Чем больше ссылок на страницу, тем она важнее

Как представляются графы

Задача вычисления на узлах
Задача обхода графа.

Как представить графы в математическом виде для MapReduce?
Как обходить граф MapReduce? – Есть информация только о части данных, и эти части не могут взаимодействовать друг с другом

Матрица смежности



	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0

Граф – матрица n * n.

Mij = 1 означает связь между i-m и j-m узлами

Матрица смежности

Плюсы:

- Простота вычислений
- Строки – исходящие ссылки, столбцы – входящие

Минусы:

- Разреженная матрица
- Требуется много места для хранения, т.к. храним нули

Список смежности

	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0

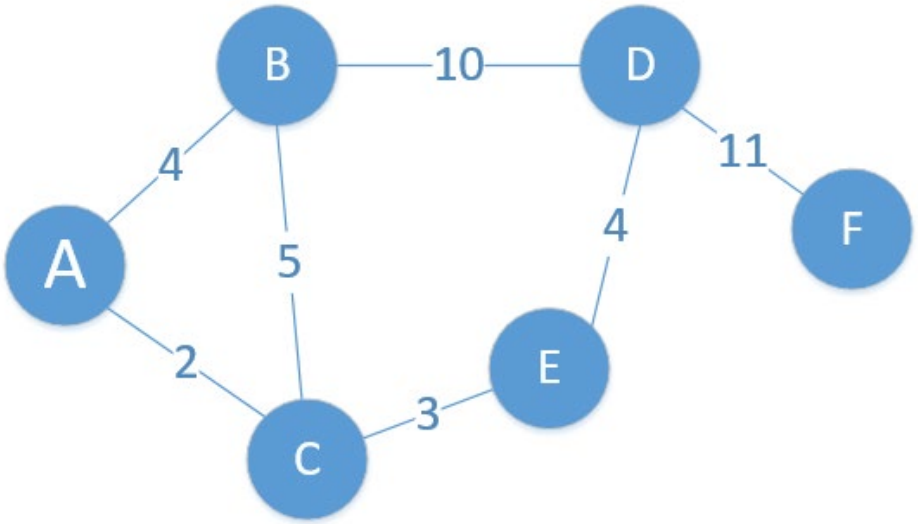
1: 2, 4
2: 1, 3, 4
3: 1
4: 1, 3

Храним только отличные от нуля значения.

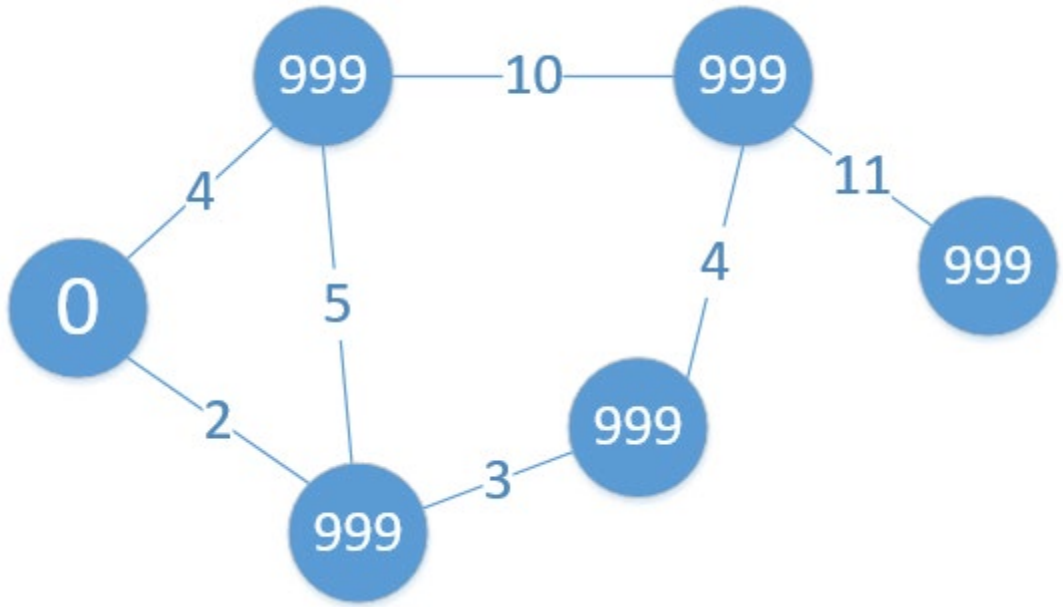
- Легко найти входящие ребра
- Сложно найти исходящие

Поиск минимального расстояния до узла

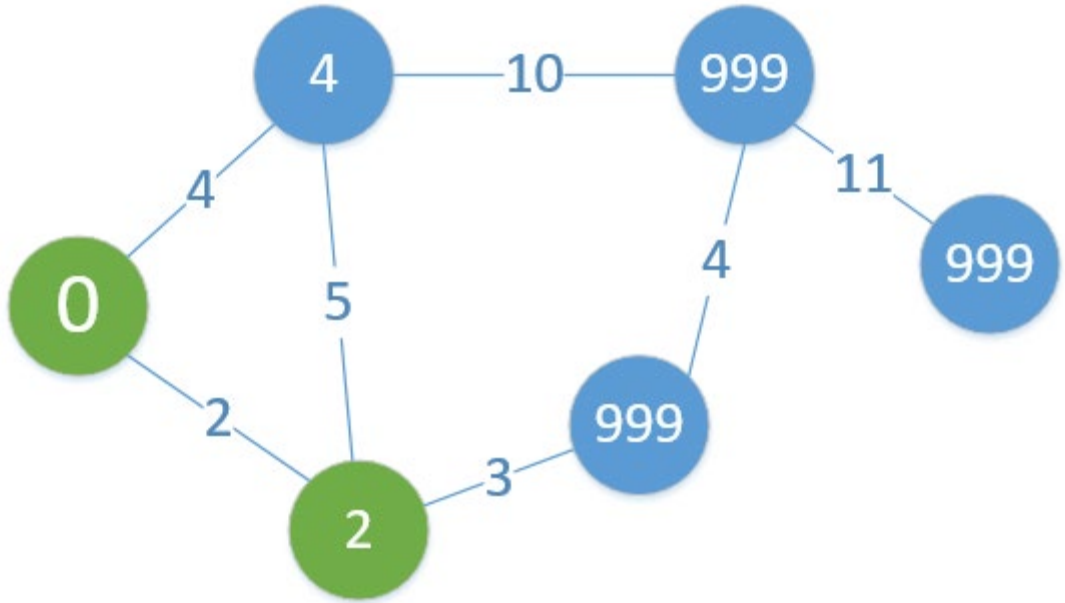
Алгоритм Дейкстры для поиска минимального расстояния



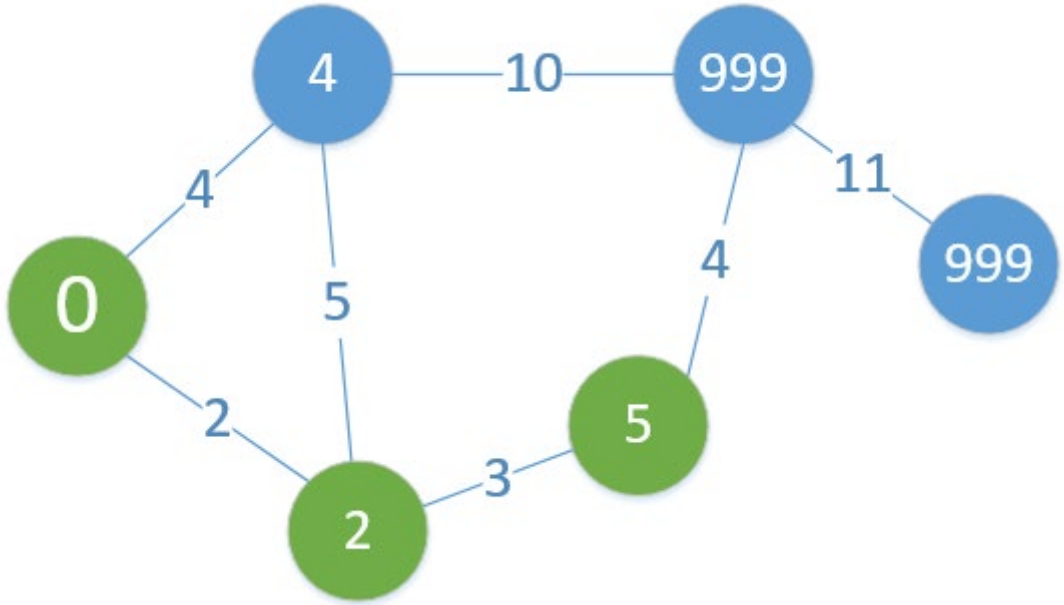
Поиск минимального расстояния до узла



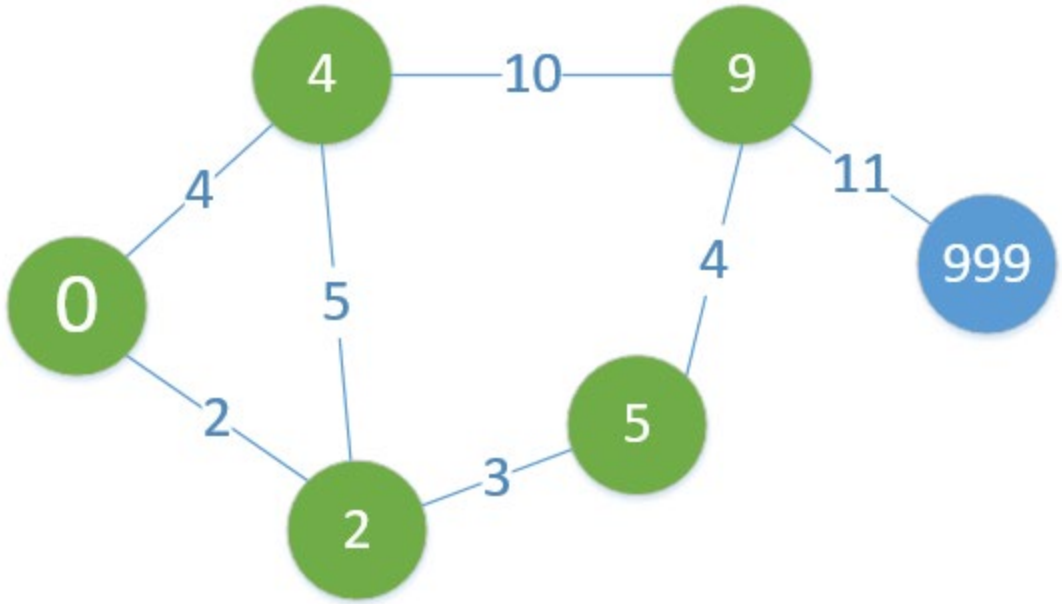
Поиск минимального расстояния до узла



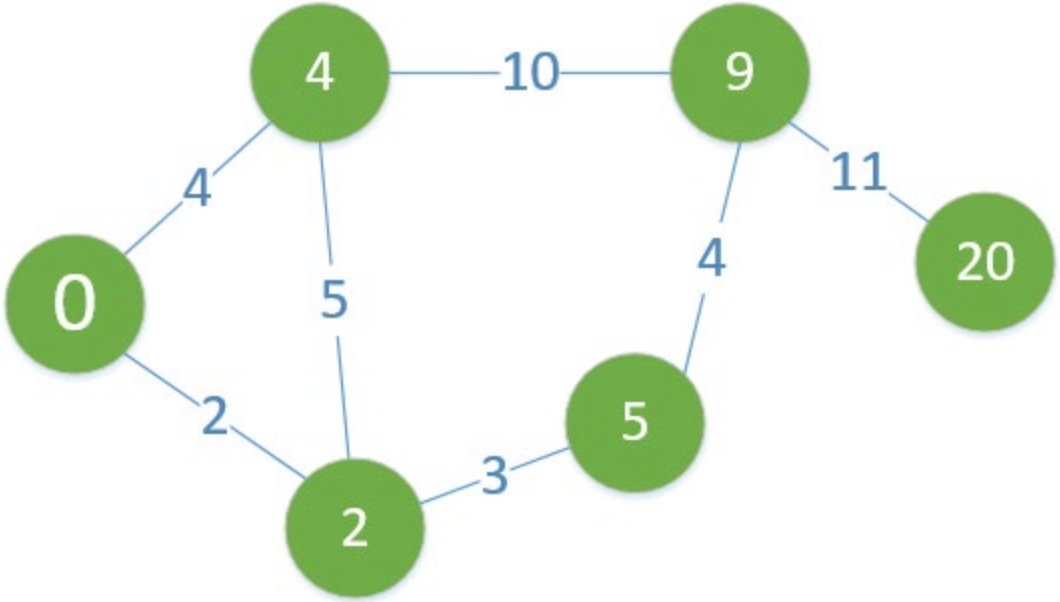
Поиск минимального расстояния до узла



Поиск минимального расстояния до узла

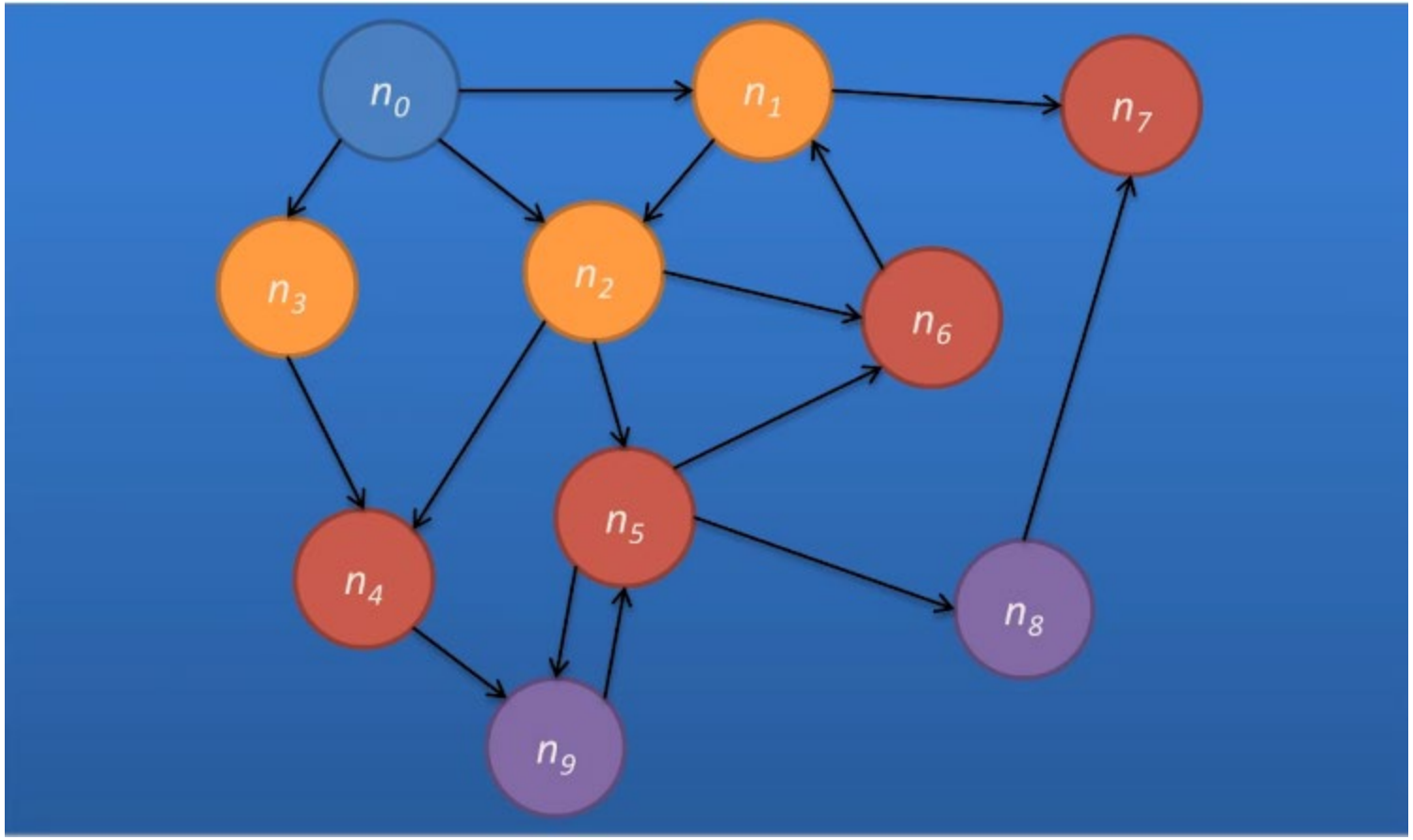


Поиск минимального расстояния до узла

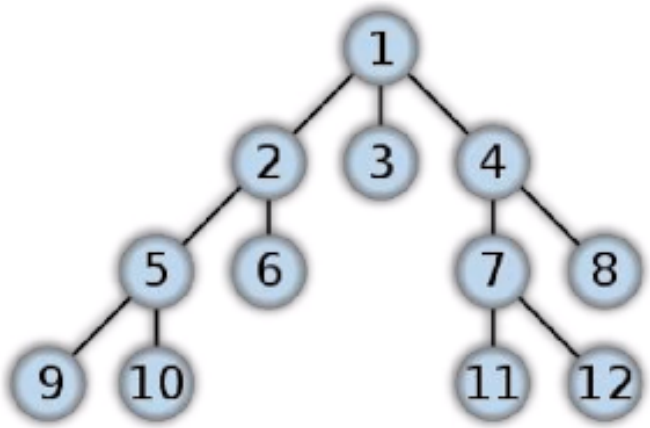


Поиск минимального расстояния до узла

Параллельный поиск в ширину (BFS, breadth-first-search)



Параллельный поиск в ширину




```
1 -> [0, {2, 3, 4}]
2 -> [∞, {5, 6} ]
3 -> [∞, {} ]
4 -> [∞, {7, 8} ]
5 -> [∞, {9, 10} ]
…
```

- Key: вершина *n*
- Value: *d* (расстояние от начала), adjacency list (вершины, доступные из *n*)
- Инициализация: для всех вершин, кроме начальной, $d = \infty$

Параллельный поиск в ширину

```
mapper(key, value):
  emit(key, value)
  ∀m ∈ value.adjacency_list: emit (m, value.d + 1)
```

Mapper 1

```
1 -> [0, {2, 3, 4}]
```

```
1 -> [0, {2, 3, 4}]
2 -> [1, {} ]
3 -> [1, {} ]
4 -> [1, {} ]
```

Mapper 2

```
2 -> [∞, {5, 6} ]
```

```
2 -> [∞, {5, 6}]
5 -> [∞, {} ]
6 -> [∞, {} ]
```

Параллельный поиск в ширину

Reduce In:

```
2 -> {[1, {} ], [∞, {5, 6}]}
```

Reduce Out:

```
2 -> [1, {5, 6} ]
```

Параллельный поиск в ширину

```
class Reducer
    method Reduce(nid m, [d1, d2, . . .])
        dmin ← ∞
        M ← Ø
        for all d ∈ counts [d1, d2, . . .] do
            if IsNode(d) then
                M ← d   // Recover graph structure
            else if d < dmin then
                dmin ← d
        M.Distance ← dmin   // Update shortest distance
        Emit(nid m, node M)
```