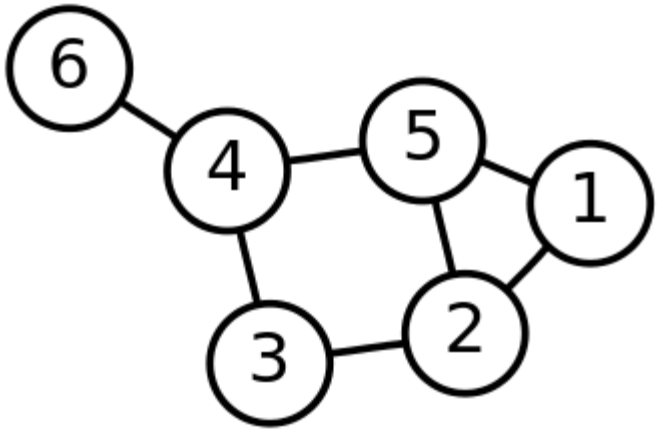
Алгоритмизация и программирование

6.5. Графы

Глухих Михаил Игоревич
mailto: glukhikh@mail.ru

Граф

- Граф = вершины (узлы) + рёбра (дуги)

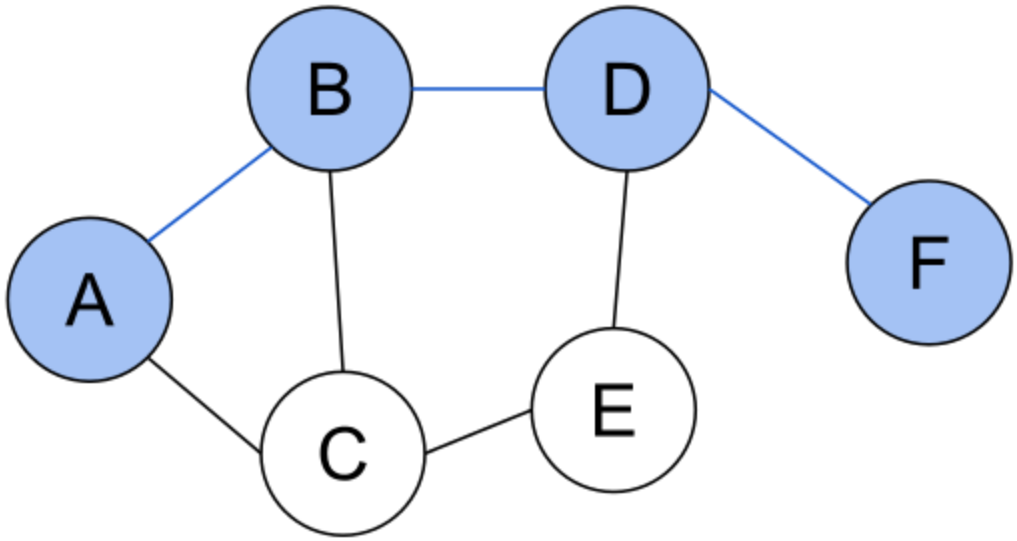


Применение графов в программировании

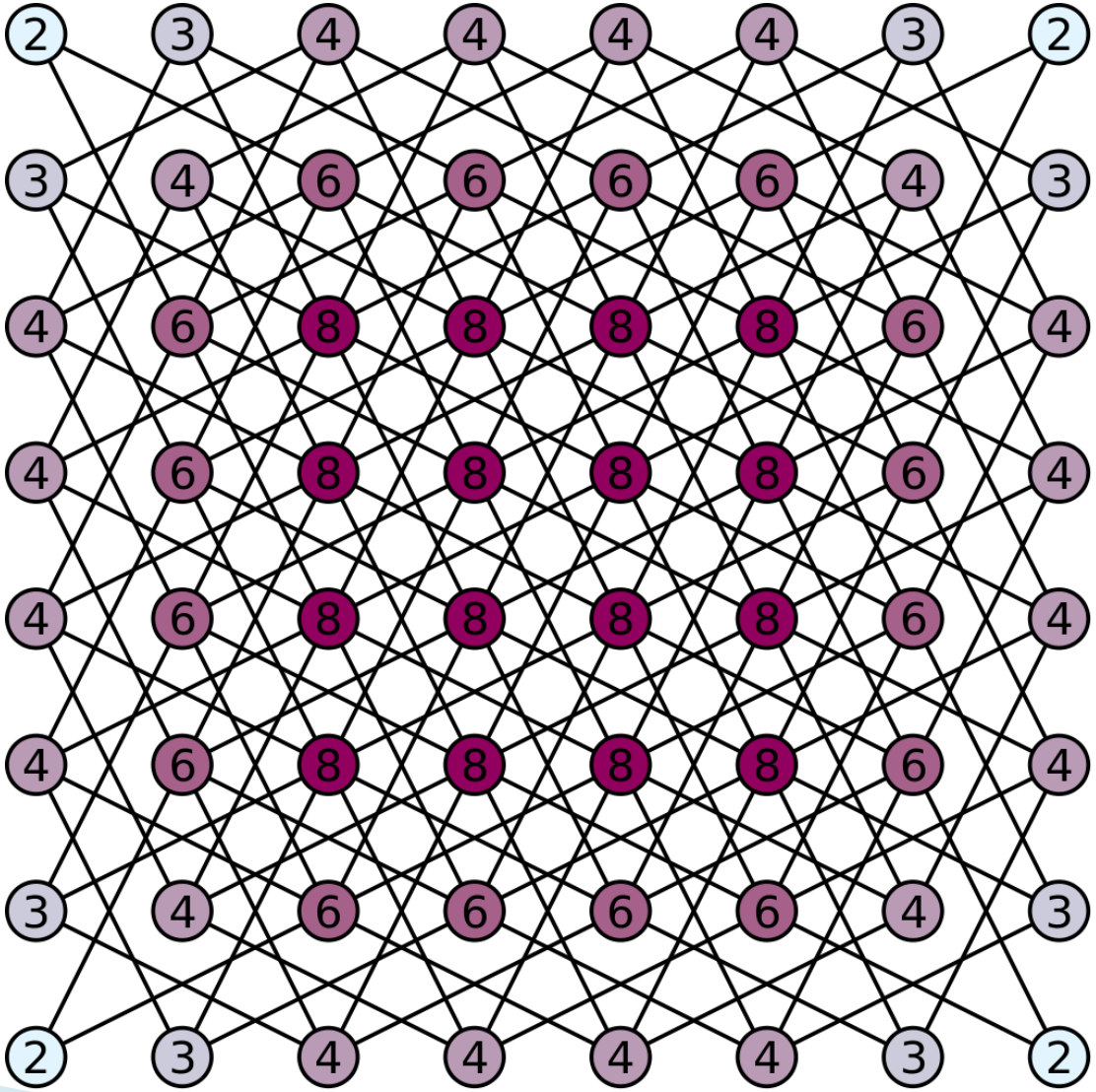
- Схемы, связи, иерархии, карты, ...
  - сети автомобильных дорог
  - схемы метро
  - компьютерные сети
  - логические схемы
  - схемы лабиринтов
  - карты дорог
  - ...

Типичная задача на графе

- Определение расстояния между вершинами



Граф на шахматной доске



Граф на Котлине

```
class Graph {
    private data class Vertex(val name: String) {
        val neighbors = mutableSetOf<Vertex>()
    }
    private val vertices = mutableMapOf<String, Vertex>()
    private fun connect(first: Vertex?, second: Vertex?) {
        if (second != null) first?.neighbors?.add(second)
        if (first != null) second?.neighbors?.add(first)
    }
    fun addVertex(name: String) {
        vertices[name] = Vertex(name)
    }
    fun connect(first: String, second: String) =
        connect(vertices[first], vertices[second])
    fun neighbors(name: String): List<String> =
        vertices[name]?.neighbors?.map { it.name } ?: listOf()
}
```

Принципы ООП

- ООП = Объектно-ориентированное программирование
- Абстракция
  - Ненужная информация опускается при описании

Принципы ООП

- ООП = Объектно-ориентированное программирование
- Абстракция
  - Ненужная информация опускается при описании
  - Ненужная информация опускается при использовании

Принципы ООП

- ООП = Объектно-ориентированное программирование
- Абстракция
  - Ненужная информация опускается при описании
  - Ненужная информация опускается при использовании
- Инкапсуляция
  - Объект = Данные + Функции (Методы)
  - Часть содержимого не видна снаружи (private)

Принципы ООП

- ООП = Объектно-ориентированное программирование
- Абстракция
  - Ненужная информация опускается при описании
  - Ненужная информация опускается при использовании
- Инкапсуляция
  - Объект = Данные + Функции (Методы)
  - Часть содержимого не видна снаружи (private)
- Наследование

Члены класса

- Состав
  - Свойства val / var
  - Функции fun
  - ~ Вложенные классы class ~

Члены класса

- Состав
  - Свойства val / var
  - Функции fun
  - ~ Вложенные классы class ~
- Видимость
  - Открытая (public) по умолчанию
  - Закрытая (private)
- Принцип разграничения ответственности

Пример использования

```
fun useGraph() {
    val g = Graph()
    g.addVertex("A")
    g.addVertex("B")
    g.addVertex("C")
    g.addVertex("D")
    g.connect("A", "C")
    g.connect("B", "D")
    g.connect("B", "C")
    println(g.neighbors("B"))
}
```

Вложенный класс

- Вершина (Vertex) не существует без графа (Graph)

Вложенный класс

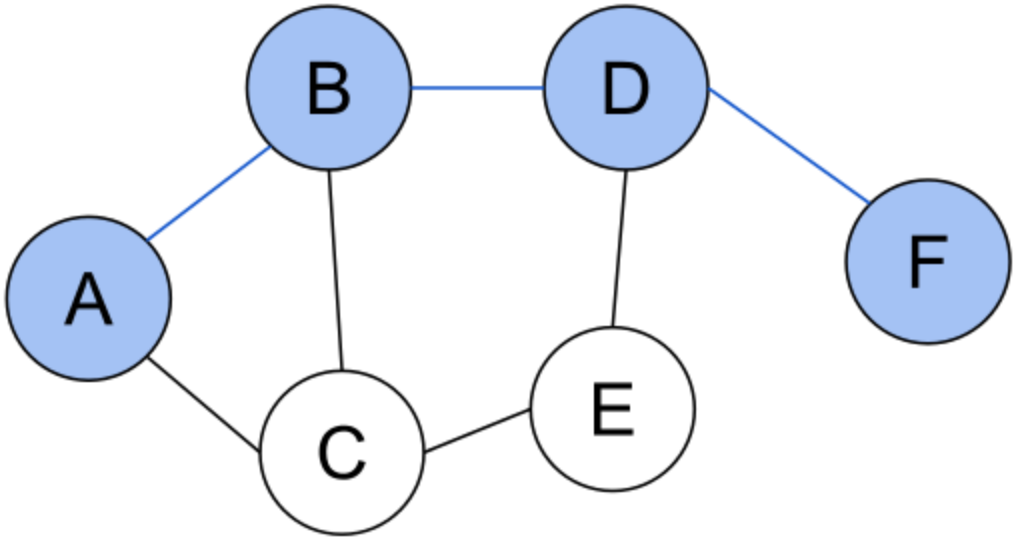
- Вершина (Vertex) не существует без графа (Graph)
- Вершина = Имя + Соседи

Вложенный класс

- Вершина (Vertex) не существует без графа (Graph)
- Вершина = Имя + Соседи
- Карта (map): Имя вершины → Вершина

Поиск на графе

- В глубину (Depth-First Search)
- В ширину (Breadth-First Search)



Поиск в глубину

- «Наивная» реализация

```
fun Graph.dfs(start: String, finish: String): Int? =
        if (start == finish) 0
        else {
            val min = neighbors(start).map {
                dfs(it, finish)
            }.filterNotNull().min()
            if (min == null) null else min + 1
        }
```

Поиск в глубину

- Полноценная реализация

```
fun Graph.dfs(start: String, finish: String) =
        dfs(start, finish, setOf()) ?: -1

fun Graph.dfs(start: String, finish: String,
              visited: Set<String>): Int? =
        if (start == finish) 0
        else {
            val min = neighbors(start).filter {
                it !in visited
            }.map {
                dfs(it, finish, visited + start)
            }.filterNotNull().min()
            if (min == null) null else min + 1
        }
```

Поиск в ширину

```
fun Graph.bfs(start: String, finish: String): Int {
    val queue = ArrayDeque<String>()
    queue.add(start)
    val visited = mutableMapOf(start to 0)
    while (queue.isNotEmpty()) {
        val next = queue.poll()
        val distance = visited[next]!!
        if (next == finish) return distance
        for (neighbor in neighbors(next)) {
            if (neighbor in visited) continue
            visited[neighbor] = distance + 1
            queue.add(neighbor)
        }
    }
    return -1
}
```

Очередь

- FIFO = First Input First Output

- poll() – взять первый элемент
- add() – добавить последний элемент

Упражнения к лекции

- Lesson6.task2 –
  две задачи про шахматного коня