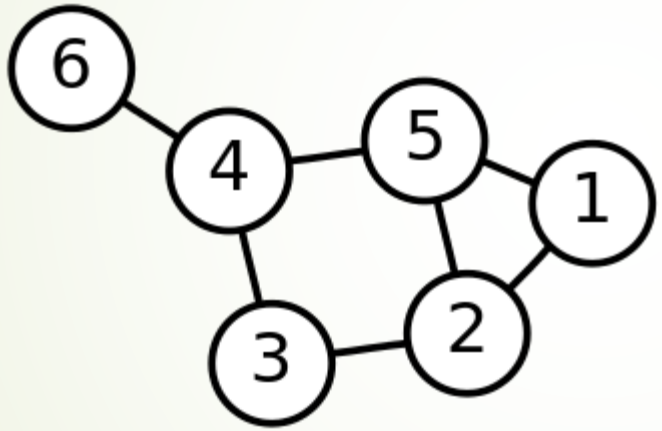
Алгоритмы и структуры данных

Лекция 3. Графы.

(c) Глухих Михаил Игоревич, glukhikh@mail.ru

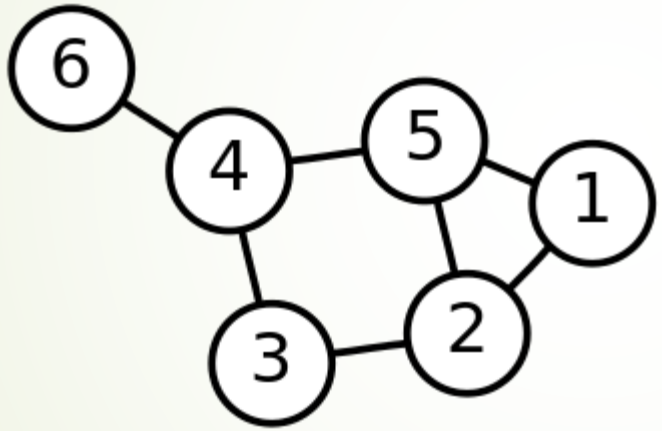
Граф

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



Граф

- Граф = вершины (узлы) + рёбра (дуги)
- Вершины и рёбра могут иметь свойства

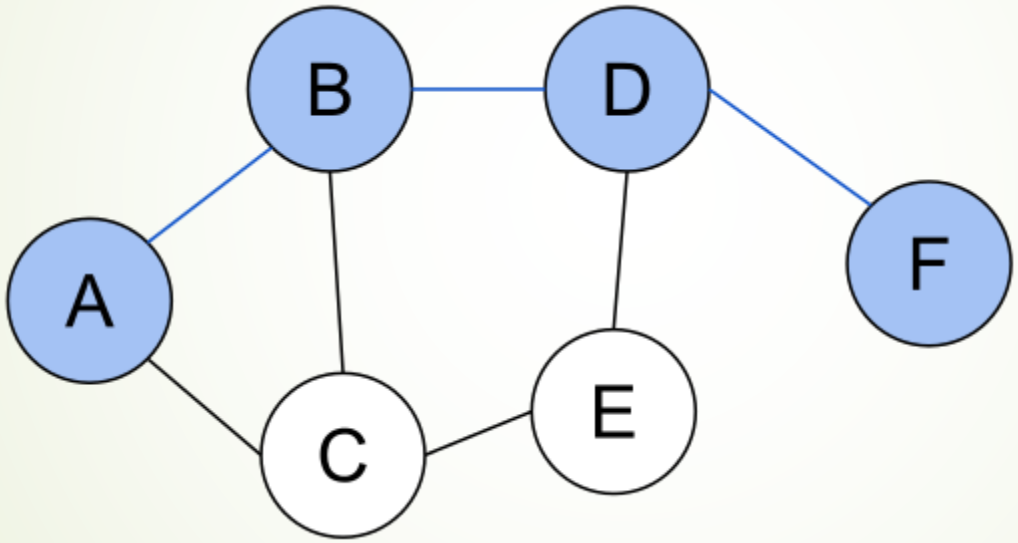


Применение графов в программировании

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

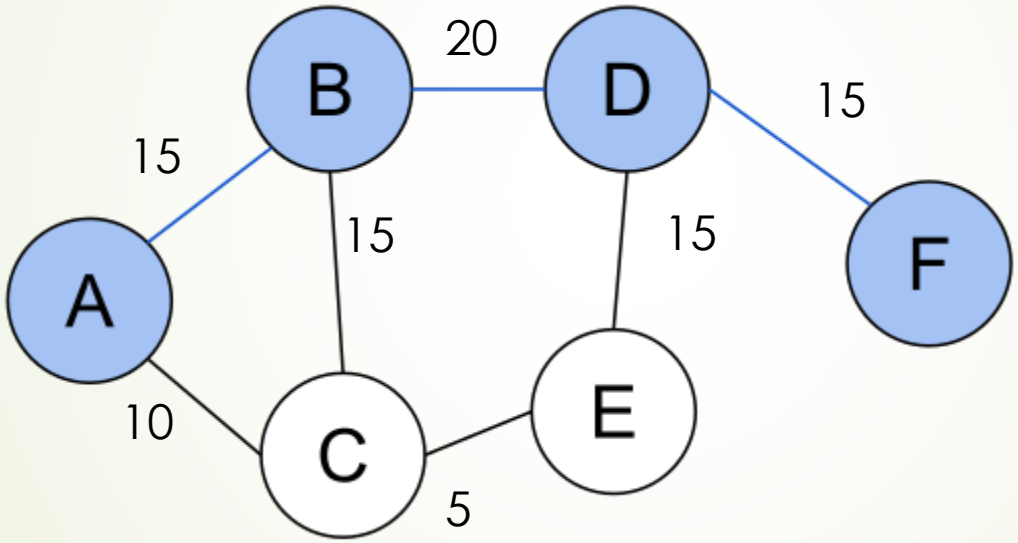
Типичная задача на графе

- Определение расстояния между вершинами

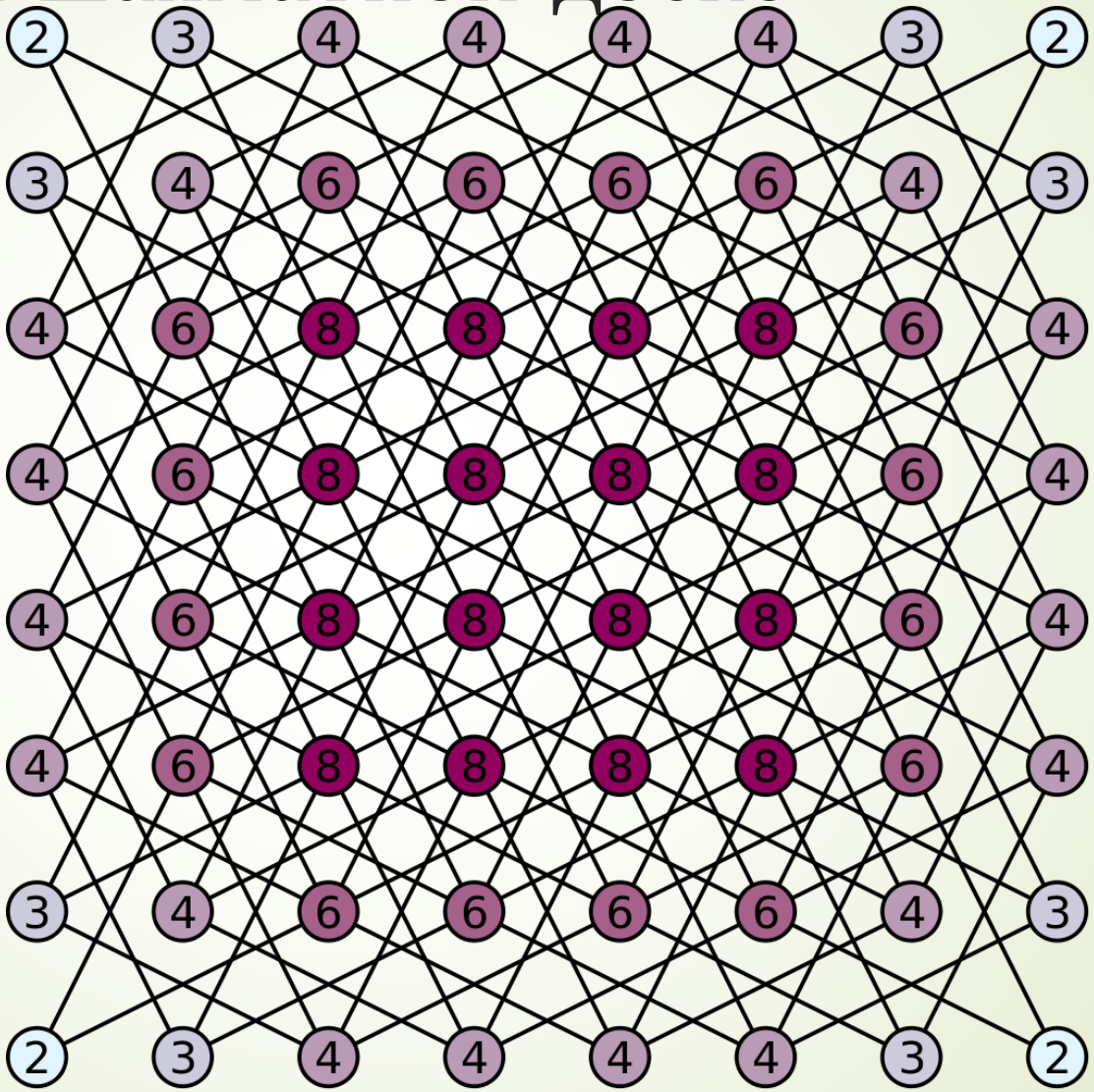


Типичная задача на графе

- Определение расстояния между вершинами



Граф на шахматной доске



Разновидности графов

- Взвешенный (с весом рёбер)

Разновидности графов

- Взвешенный (с весом рёбер)
- Ориентированный / неориентированный

Разновидности графов

- Взвешенный (с весом рёбер)
- Ориентированный / неориентированный
- Мультиграф (с кратными рёбрами)

Разновидности графов

- Взвешенный (с весом рёбер)
- Ориентированный / неориентированный
- Мультиграф (с кратными рёбрами)
- Дерево (без циклов)

Представление графа

- В виде списка смежных вершин

Представление графа

- В виде списка смежных вершин
  - То есть, для каждой вершины храним список соседей (и сопутствующую информацию)

Интерфейс «Граф» на Java (пример)

```
public interface Graph {
    interface Vertex {
        String getName();
    }
    Set<Vertex> getVertices();
    Set<Vertex> getNeighbors(Vertex v);
    // Optional, for weighted graph
    interface Edge {
        int getWeight();
    }
    Map<Vertex, Edge> getConnections(Vertex v);
}
```

Представление графа

- В виде списка смежных вершин
  - То есть, для каждой вершины храним список соседей (и сопутствующую информацию)
- В виде матрицы смежности

Представление графа

- В виде списка смежных вершин
  - То есть, для каждой вершины храним список соседей (и сопутствующую информацию)
- В виде матрицы смежности
  - Строки и столбцы = вершины, ячейки = дуги

Представление графа

- В виде списка смежных вершин
  - То есть, для каждой вершины храним список соседей (и сопутствующую информацию)
- В виде матрицы смежности
  - Строки и столбцы = вершины, ячейки = дуги
- Что требует больше места?

Представление графа

- В виде списка смежных вершин $O(V) + O(E)$
  - То есть, для каждой вершины храним список соседей (и сопутствующую информацию)
- В виде матрицы смежности $O(V^2)$
  - Строки и столбцы = вершины, ячейки = дуги
- Что требует больше места?

Поиск пути на графе

- Поиск в ширину (BFS, Breadth-First Search)

Поиск пути на графе

- Поиск в ширину (BFS, Breadth-First Search)
  - Проверяем вершины последовательно по возрастанию пути до них

Поиск пути на графе

- Поиск в ширину (BFS, Breadth-First Search)
  - Проверяем вершины последовательно по возрастанию пути до них
- Поиск в глубину (DFS, Depth-First Search)

Поиск пути на графе

- Поиск в ширину (BFS, Breadth-First Search)
  - Проверяем вершины последовательно по возрастанию пути до них
- Поиск в глубину (DFS, Depth-First Search)
  - Проверяем каждый путь, пока не встретим тупик / кольцо

Поиск в ширину: псевдокод

```
BFS(G = (V, E), s in V):
  for (v in V):
    info[v] = (visit = NOT_VISITED, prev = null)
  info[s] = (VISITED, null)
  ENQUEUE(s)
  while (QUEUE is not empty):
    u = DEQUEUE()
    for (v in neighbors(u)):
      if (info[v] NOT_VISITED):
        info[v] = (VISITED, prev = u)
        ENQUEUE(v)
```

Поиск в ширину: трудоёмкость

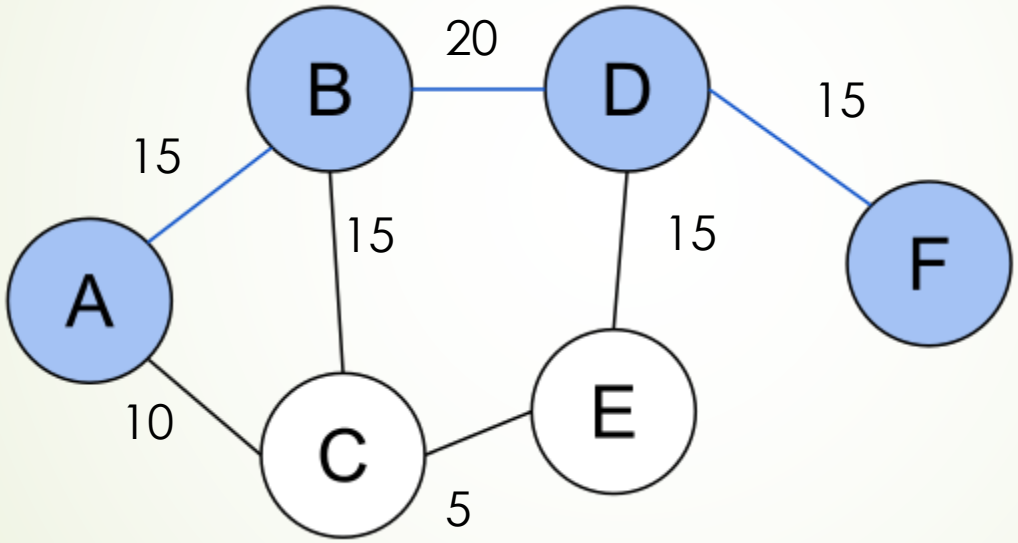
```
BFS(G = (V, E), s in V):
  for (v in V):
    info[v] = (visit = NOT_VISITED, prev = null)
  info[s] = (VISITED, null)
  ENQUEUE(s)
  while (QUEUE is not empty):
    u = DEQUEUE() // V iterations
    for (v in neighbors(u)):
      if (info[v] NOT_VISITED): // E iterations
        info[v] = (VISITED, prev = u)
        ENQUEUE(v)
```

Поиск пути в ширину: псевдокод

```
BFS(G = (V, E), s in V):
  for (v in V):
    info[v] = (visit = NOT_VISITED, distance = INF, prev = null)
  info[s] = (VISITED, 0, null)
  ENQUEUE(s)
  while (QUEUE is not empty):
    u = DEQUEUE()
    for (v in neighbors(u)):
      if (info[v] NOT_VISITED):
        info[v] = (VISITED, distance = info[u].distance + 1, prev = u)
        ENQUEUE(v)
```

Поиск в ширину: что меняется, если граф взвешенный?

Поиск в ширину: что меняется, если граф взвешенный?



Поиск в ширину: алгоритм Дейкстры

```
BFS(G = (V, E), s in V):
  for (v in V):
    info[v] = (distance = INF, prev = null)
  info[s] = (0, null)
  ENQUEUE(s)
  while (QUEUE is not empty):
    u = DEQUEUE() // with shortest distance (greedy)
    for (v in neighbors(u))
      if (info[v].distance > info[u].distance + distance(u, v)):
        info[v] = (info[u].distance + distance(u, v), prev = u)
        ENQUEUE(v) // replacing previous (v) if necessary
```

Алгоритм Дейкстры: трудоёмкость

```
BFS(G = (V, E), s in V):
  for (v in V):
    info[v] = (distance = INF, prev = null)
  info[s] = (INF, null)
  ENQUEUE(s)
  while (QUEUE is not empty):
    u = DEQUEUE() // V iterations
    for (v in neighbors(u))
      if (info[v].distance > info[u].distance + 1): // E iterations
        info[v] = (distance = info[u].distance + 1, prev = u)
        ENQUEUE(v)
```

Алгоритм Дейкстры: трудоёмкость

```
BFS(G = (V, E), s in V):
  for (v in V):
    info[v] = (distance = INF, prev = null)
  info[s] = (INF, null)
  ENQUEUE(s)
  while (QUEUE is not empty):
    u = DEQUEUE() // V iterations, ~ log(V) each
    for (v in neighbors(u))
      if (info[v].distance > info[u].distance + 1): // E iterations
        info[v] = (distance = info[u].distance + 1, prev = u)
        ENQUEUE(v)
```

Поиск в глубину: псевдокод

```
DFS(G = (V, E), s in V):
  for (v in V):
    info[v] = (NOT_VISITED, prev = null)
  DFS-VISIT(G, s, info, prev = null)

DFS-VISIT(G = (V, E), v in V, info, prev):
  info[v] = (VISITED, prev = prev)
  for (u in neighbors(v)):
    if (info[u] NOT_VISITED):
      DFS-VISIT(G, u, info, prev = v)
```

Поиск в глубину: трудоёмкость

```
DFS(G = (V, E), s in V):
  for (v in V):
    info[v] = (NOT_VISITED, prev = null)
  DFS-VISIT(G, s, info, prev = null)

DFS-VISIT(G = (V, E), v in V, info, prev): // V calls
  info[v] = (VISITED, prev = prev)
  for (u in neighbors(v)):
    if (info[u] NOT_VISITED): // Total E iterations
      DFS-VISIT(G, u, info, prev = v)
```

Поиск пути в глубину: псевдокод

```
DFS(G = (V, E), s in V):
  for (v in V):
    info[v] = (distance = INF, prev = null)
  DFS-VISIT(G, s, info, depth = 0, prev = null)

DFS-VISIT(G = (V, E), v in V, info, depth, prev):
  info[v] = (distance = depth, prev = prev)
  for (u in neighbors(v)):
    if (info[u].distance > depth + 1):
      DFS-VISIT(G, u, info, depth = depth + 1, prev = v)
```

Поиск пути: что лучше?

- В ширину / В глубину?

Поиск пути: что лучше?

- В ширину / В глубину?
  - Для невзвешенного графа ~ без разницы: O(V) + O(E)

Поиск пути: что лучше?

- В ширину / В глубину?
  - Для невзвешенного графа ~ без разницы: O(V) + O(E)
  - Для взвешенного графа алгоритм Дейкстры даёт O(VlogV)+ O(E)

Поиск пути: что лучше?

- В ширину / В глубину?
  - Для невзвешенного графа ~ без разницы: O(V) + O(E)
  - Для взвешенного графа алгоритм Дейкстры даёт O(VlogV)+ O(E)
  - Что может дать поиск в глубину?

Поиск пути: что лучше?

- В ширину / В глубину?
  - Для невзвешенного графа ~ без разницы: O(V) + O(E)
  - Для взвешенного графа алгоритм Дейкстры даёт O(VlogV)+ O(E)
  - Что может дать поиск в глубину?
    - Нам придётся смотреть одни и те же вершины / рёбра несколько раз…

Применение поиска в глубину

- Перебор вариантов в логических играх
  - (с ограничением глубины)
- Топологическая сортировка
  - Упорядочение вершин ориентированного графа

Задача коммивояжёра

- Поиск кратчайшего пути, посещающего все вершины и возвращающегося в исходную
- Классический пример задачи, для которой нет «быстрого» решения

Задача коммивояжёра: точное решение

- Используем поиск в глубину (отбрасывая вершины, уже вошедшие в данный путь)
- Запоминаем самый короткий маршрут
- Отсечение: останавливаем текущую ветку поиска в глубину, если её длина уже превысила длину самого короткого маршрута

Итоги

- Рассмотрено
  - Понятие графа
  - Разновидности графов
  - Поиск в ширину / глубину
  - Алгоритм Дейкстры
- Далее
  - Таблицы и деревья