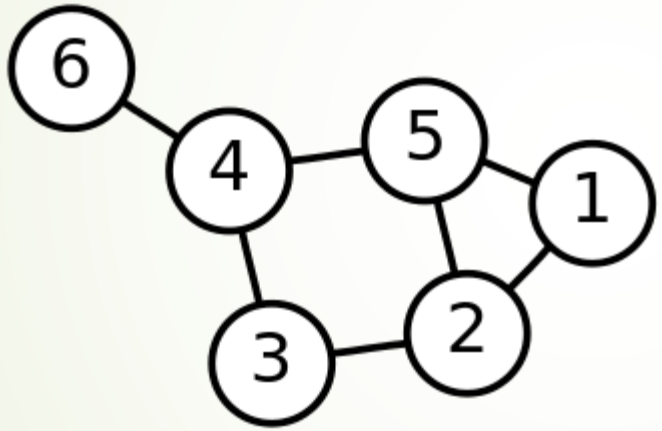
Алгоритмы и структуры данных

Лекция 7. Графы (поиск кратчайшего пути).

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

Граф

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

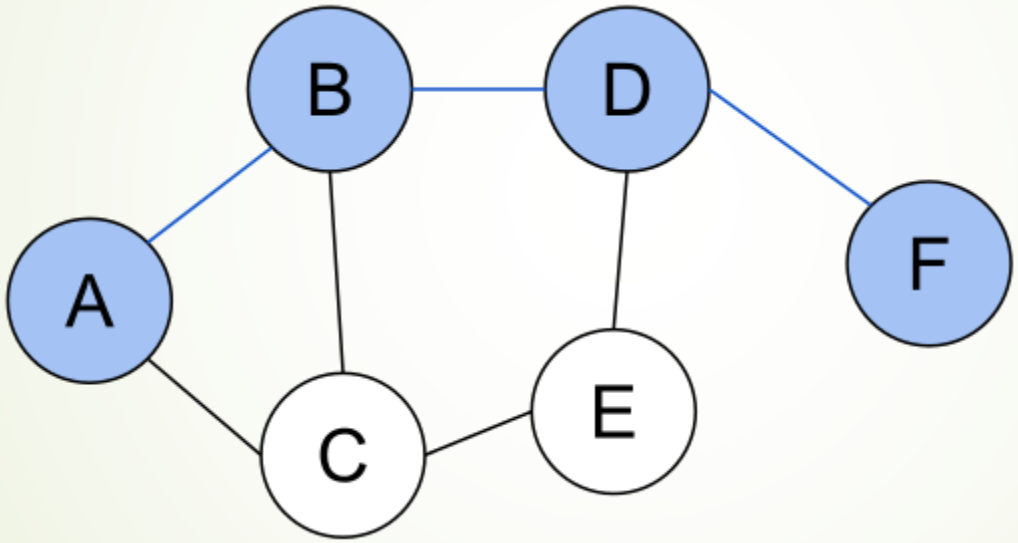


Применение графов в программировании

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

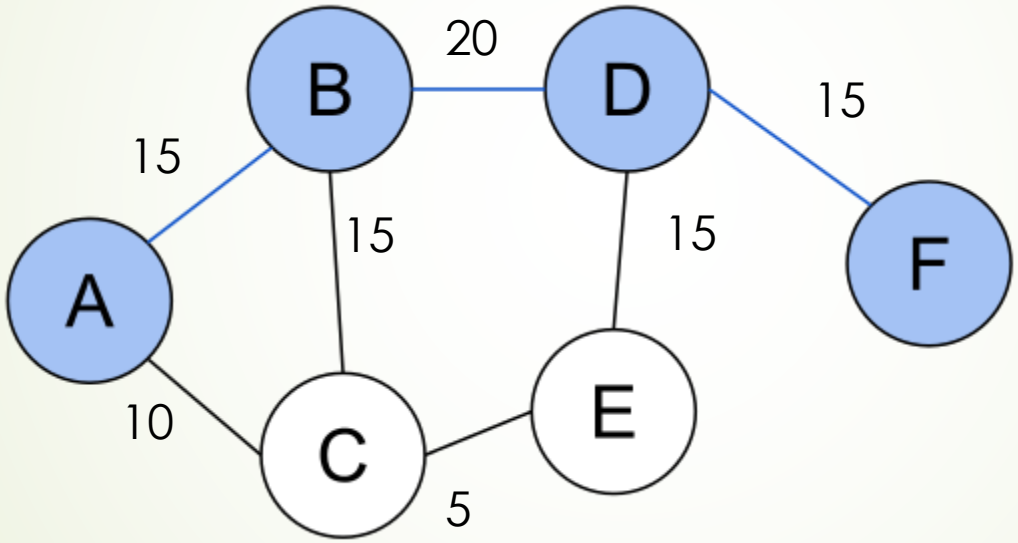
Типичная задача на графе

- Определение расстояния между вершинами



Типичная задача на графе

- Определение расстояния между вершинами



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)

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?

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?
  - Что всё же выгоднее?

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?
  - Что всё же выгоднее?
    - $0 <= E <= V^2$

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);
}
```

Обход графа

- 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

Обход графа

- 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
- NB: искать мы можем, в общем случае, что угодно!

Обход графа

- 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
- NB: искать мы можем, в общем случае, что угодно!
  - Пример (лишь пример!): поиск пути в графе

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].visit = NOT_VISITED):
        info[v] = (VISITED, prev = u)
        ENQUEUE(v)
```

Поиск в ширину: трудоёмкость

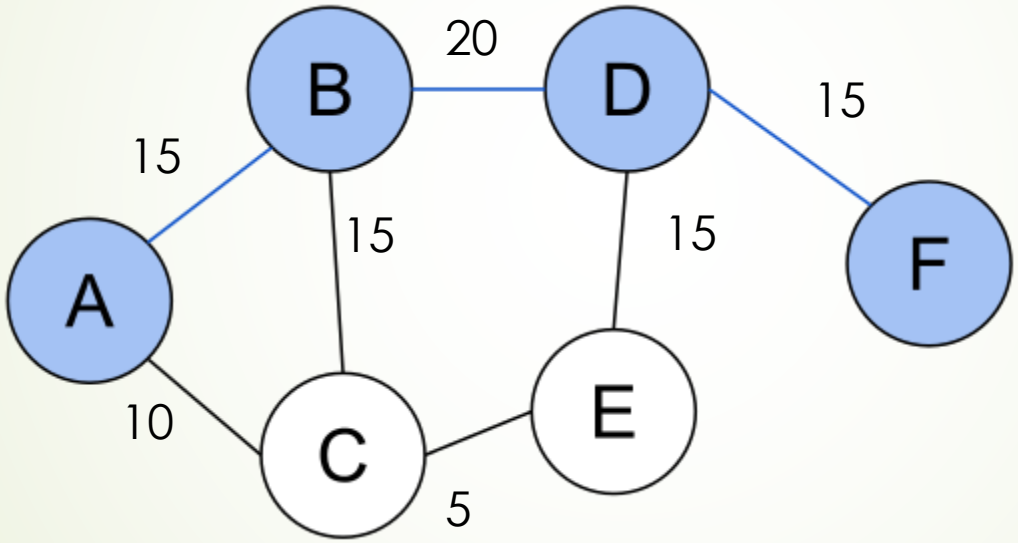
```
BFS(G = (V, E), s in V):
  for (v in V):
    info[v] = (visit = NOT_VISITED, prev = null)
  info[s] = (VISITED, null)
  ENQUEUE(s)
  while (QUEUE is not empty):
    u = DEQUEUE() // V iterations
    for (v in neighbors(u)):
      if (info[v].visit = NOT_VISITED): // E iterations
        info[v] =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

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, distance = 0, null)
  ENQUEUE(s)
  while (QUEUE is not empty):
    u = DEQUEUE()
    for (v in neighbors(u)):
      if (info[v].visit = 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); visited = ()
  ENQUEUE(s)
  while (QUEUE is not empty):
    u = DEQUEUE() // with shortest distance (greedy)
    visited.add(u)
    for (v in neighbors_not_visited(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
    visited.add(u)
    for (v in neighbors_not_visited(u)) // E iterations
      if (info[v].distance > info[u].distance + distance(u, v)):
        info[v] = (info[u].distance + distance(u, v), 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 (heap in use)
    visited.add(u)
    for (v in neighbors_not_visited(u)) // E iterations
      if (info[v].distance > info[u].distance + distance(u, v)):
        info[v] = (info[u].distance + distance(u, v), prev = u)
        ENQUEUE(v) // ~ log(V) each
// Total: (V+E)logV
```

Очередь с приоритетами

- (with shortest distance…)
- Java: PriorityQueue
- Вариант реализации: binary heap

Очередь с приоритетами

- (with shortest distance…)
- Java: PriorityQueue
- Вариант реализации: binary heap (двоичная куча)
  - Вершина с минимальной дистанцией хранится на вершине кучи
  - Добавление и удаление – за O(logV)

Волновой алгоритм

- Он же алгоритм Ли
- = Поиск пути на **планарном** графе на **основе** поиска в **ширину**

Волновой алгоритм

- Он же алгоритм Ли
- = Поиск пути на **планарном** графе
  на **основе** поиска в **ширину**
  - Планарный граф = тот, который можно нарисовать на плоскости без пересечений рёбер

Волновой алгоритм

- Он же алгоритм Ли
- = Поиск пути на **планарном** графе
  на **основе** поиска в **ширину**
  - Планарный граф = тот, который можно нарисовать на плоскости без пересечений рёбер
- Трудоёмкость = обычный поиск пути в ширину
  (на графе без весов)

Волновой алгоритм

9	10		10	9	8	9	10	11	12	13	14
8	9		9	8	7	8	9	10	11	12	13
7	8	9	8	7	6	7	8	9	10	11	12
6	7	8	7	6	5	6	7			10	11
5					4	5	6	7	8	9	10
4	3	2	1	2	3	4	5	6			11
3	2	1	0	1	2	3	4	5			10
4	3	2	1	2	3	4	5	6	7	8	9

Алгоритм A-Star (A*)

- Поиск кратчайшего пути, улучшение алгоритма Дейкстры
  - Даёт возможность посетить меньше вершин

Алгоритм A-Star (A*)

- Поиск кратчайшего пути, улучшение алгоритма Дейкстры
  - Даёт возможность посетить меньше вершин
- Отличие: в порядке обхода вершин
  - Дейкстра: критерием является расстояние от старта

Алгоритм A-Star (A*)

- Поиск кратчайшего пути, улучшение алгоритма Дейкстры
  - Даёт возможность посетить меньше вершин
- Отличие: в порядке обхода вершин
  - Дейкстра: критерием является расстояние от старта
  - A*: критерием является расстояние до старта + расстояние до финиша

Алгоритм A-Star (A*)

- Поиск кратчайшего пути, улучшение алгоритма Дейкстры
  - Даёт возможность посетить меньше вершин
- Отличие: в порядке обхода вершин
  - Дейкстра: критерием является расстояние от старта
  - A*: критерием является расстояние до старта + расстояние до финиша
  - ?! Его же ещё надо найти ?!

Алгоритм A-Star (A*)

- Поиск кратчайшего пути, улучшение алгоритма Дейкстры
  - Даёт возможность посетить меньше вершин
- Отличие: в порядке обхода вершин
  - Дейкстра: критерием является расстояние от старта
  - A*: критерием является расстояние до старта + расстояние до финиша
  - ?! Его же ещё надо найти ?!
    - Рассчитанное эвристически…

Алгоритм A-Star (A*)

- Поиск кратчайшего пути, улучшение алгоритма Дейкстры
  - Даёт возможность посетить меньше вершин
- Отличие: в порядке обхода вершин
  - Дейкстра: критерием является расстояние от старта
  - A*: критерием является расстояние до старта + расстояние до финиша
  - ?! Его же ещё надо найти ?!
    - Рассчитанное эвристически…
    - На прямоугольном поле: (разница по ширине + разница по высоте)

Алгоритм A-Star (A*)

- Эвристика: оценка расстояния от текущей вершины до конечной вершины
- Требования
  - Оценка снизу (реальное расстояние >= оценки)
  - Неравенство треугольника
- Иначе можем пропустить оптимальное решение

Алгоритм Беллмана-Форда

- Тоже ищет кратчайшие пути в графе
- Допускает рёбра с отрицательным весом (но в графе не должно быть отрицательных циклов)
- O(VE)
- Перебор всех рёбер V-1 раз с обновлением оптимальных расстояний, если они есть

Итоги

- Рассмотрено
  - Граф и способы его представления
  - Поиск в ширину, алгоритмы поиска кратчайшего пути
- Далее
  - Поиск в глубину, поиск наилучшего хода, …