Algorithms With Java

Week 1 Day 2-3: Задача перколяции

Академия Ковалевского

Содержание

1. Теория **3**
Компонента связности графа 3
Алгоритм поиска компонент связности 3
Задача перколяции 3

2. Практическая работа **4**
Node 4
Graph 4
Percolation 6

1. Теория

Компонента связности графа

Граф называется связным, если между каждой парой вершин есть путь. Понятие компоненты связности вытекает из понятия связности графа. Попросту говоря, компонента связности — часть графа (подграф), являющаяся связной. Формально, компонента связности — набор вершин графа, между любой парой которых существует путь.

Алгоритм поиска компонент связности

Для поиска компонент связности используется обычный DFS практически без модификаций (можно использовать и BFS). При запуске обхода из одной вершины, он гарантированно посетит все вершины, до которых возможно добраться, то есть, всю компоненту связности, к которой принадлежит начальная вершина. Для нахождения всех компонент просто попытаемся запустить обход из каждой вершины по очереди, если мы ещё не обошли её компоненту ранее.

Задача перколяции

Для понимания физических явлений часто используют модели физических систем. Построение перколяционной модели применяется во многих областях физики и не только, например: просачивание жидкостей через пористую среду, фазовый переход изолятор-проводник, процесс распространения эпидемии или лесного пожара, построение модели социальной сети и т.д.

Краткую теорию можно найти по этой ссылке.

2. Практическая работа

Сегодня мы реализуем алгоритм подсчета компонент связностей у графа, а после посмотрим как этот алгоритм можно применить для анализа перколяции.

Node

Любая вершина графа, в сегодняшнем дне, будет описан следующем классом:

```
package academy.kovalevskyi.algorithms.week1.day2;

import java.util.HashSet;
import java.util.Set;

public class Node {
 private final Set<Node> neighbours = new HashSet<>();

 public static void connect(Node left, Node right) {
   left.neighbours.add(right);
   right.neighbours.add(left);
 }

 public Set<Node> getNeighbours() {
   return neighbours;
 }
}
```

Graph

В данному классе нужно реализовать собственно сам алгоритм построения графа. Обратите внимание, что:

- Конструктор должен быть private
- Создать новый экземпляр класса можно только вызывая статический метод

Это довольно распространенный подход когда при создании класса нужно выполнить какой-либо подсчет, в конструкторе никогда не стоит делать какие-либо сложные/долгие/IO/etc операции, а вынося процесс создания в метод, сразу понятно, что метод не только создает объект, но и делает что-то еще.

Вся логика подсчета количества компонент связности должна быть в методе generateGraph, и уже подсчитанные данные переданы в конструктор. Подумайте в каком виде лучше всего хранить данные после подсчета.

Соответственно, методы countComponents() и getComponentId() должны просто возвращать данные (а не высчитывать каждый раз).

Для метода getComponentId() не важно какой id он возвращает, важно чтобы для одной и той же компоненты связности возвращался один и тот же id.

В данном задание нет требования к сложности, однако важно верно оценить какая именно сложность у решения.

```
package academy.kovalevskyi.algorithms.week1.day2;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;

public class Graph {

// TODO private constructor

  public static Graph generateGraph(Set<Node> nodes) {
    // TO DO
  }

  public int countComponents() {
    // TO DO
  }

  public int getComponentId(Node node) {
    // TO DO
  }
}
```

Percolation

А теперь, собственно, мы применим наш класс в решении реальной задачи. Представьте, что есть труба наполненная каким-то материалом (все равно каким). На вход (слева) в трубу подается вода. Важно понять, дойдет ли вода до правой части или нет.

На вход класса подается двумерный массив, описывающий трубу. Пример входных данных:

{{ true, true, false },

{ false, true, false },

{ false, true, true }}

В данном примере труба размером 3 на 3. Если в ячейке трубы *true* — это значит, что вода может проникнуть в эту ячейку. Так как вода подается слева, то мы видим, что вода сможет проникнуть в левую верхнюю ячейку:

{{ **true**, true, false },

{ false, true, false },

{ false, true, true }}

После этого вода сможет проникнуть в ячейку справа (важно, что вода не может проникать в ячейки по диагонали):

{{ **true**, **true**, false },

{ false, true, false },

{ false, true, true }}

Продолжая мы увидим, что вода дойдет до правой части:

{{ **true**, **true**, false },

{ false, **true**, false },

{ false, **true**, **true** }}

А вот пример трубы, где вода дойти не сможет:

{ { **true**, **true**, false, true},

{ false, **true**, false, true },

{ false, **true**, false, true }}

В методе percolate() нужно при помощи использования класса **Graph** написать логику, которая благодаря анализу компонент связности графа будет возвращать *true*, если вода дойдет слева направо и *false* если нет.

```
package academy.kovalevskyi.algorithms.week1.day2;

import java.util.HashSet;
import java.util.Set;

public class Percolation {

  public Percolation(boolean[][] field) {
    // TO DO
  }

  public boolean percolate() {
    // TO DO
  }
}
```