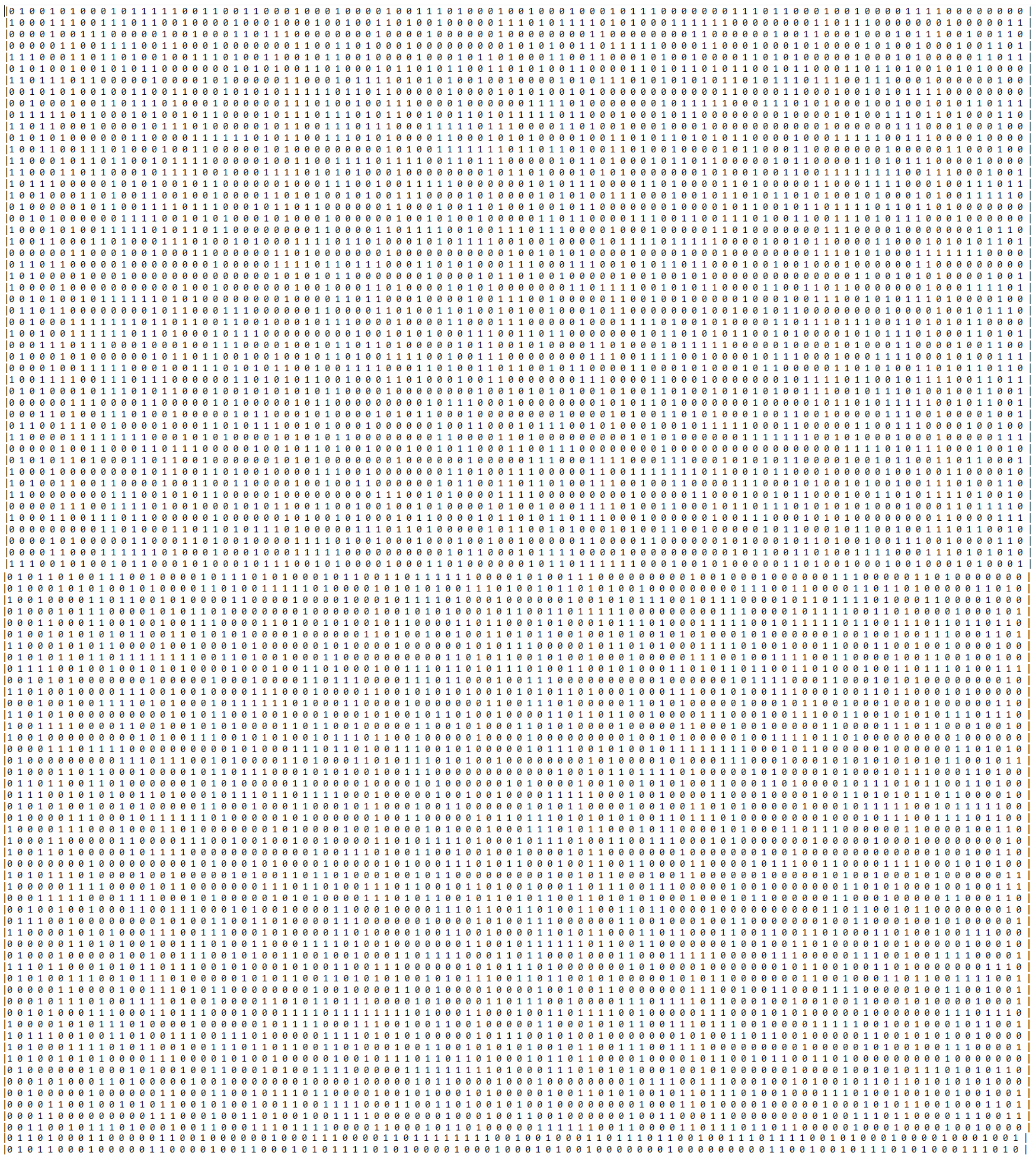
Задание 5. Алгоритмы на графах. Введение в графы и основные алгоритмы на графах

Варвара Кошман, C4113, 25.11.2019

I. Генерация случайного неориентированного простого графа с 100 вершинами и 2000 ребрами. Построение матрицы смежности и списков смежности. Визуализация построенного графа.

Матрица смежности:

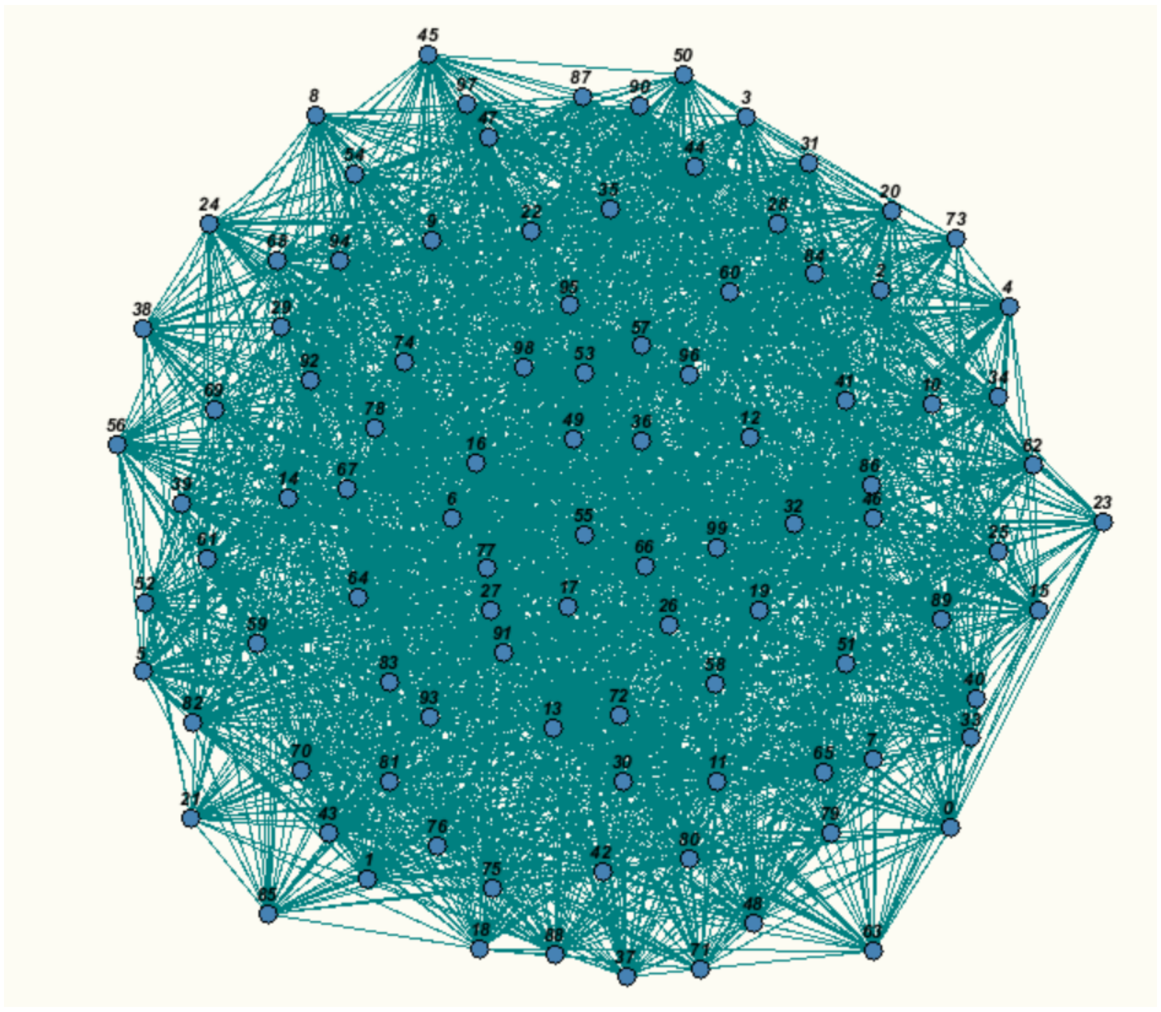


```
|0 1 0 0 1 0 1 0 0 0 1 0 1 1 1 1 1 0 0 1 1 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 1 1 1 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 1 1 1 0 1 1 0 0 0 1 0 0 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0|
|1 0 0 0 1 1 1 0 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 1 1 0 1 0 0 1 0 0 0 0 0 1 1 1 0 1 0 1 1 1 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1|
|0 0 0 0 1 0 0 1 1 1 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 1 0 1 1 1 0 0 1 0 0 1 1 0|
|0 0 0 0 0 1 1 0 0 1 1 1 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1 1 0 1 1 1 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 0 0 1 0 0 1 1 0 1|
|1 1 1 0 0 0 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 0 1 0 0 0 1 1 0 0 1 1 0 0 0 1 0 1 0 0 1 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 1 1|
|0 1 0 1 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1 1 0 1 0 0 0 1 0 1 1 0 1 0 1 1 0 0 1 1 0 1 0 1 0 0 1 1 0 0 0 0 1 1 0 1 0 1 1 0 1 0 1 1 0 0 1 0 1 1 0 0 0 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0|
|1 1 0 1 1 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1 1 1 1 0 1 0 1 0 1 0 0 1 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0 1 0 1 0 1 0 1 0 1 1 0 1 0 1 1 1 0 1 1 1 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0|
|0 0 1 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 1 0 0 1 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0|
|0 0 1 0 0 0 1 0 0 1 1 0 1 1 1 0 1 0 0 0 1 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 1 0 0 1 0 0 1 0 1 0 1 1 0 1 1 1|
|0 1 1 1 1 1 0 1 1 0 0 0 1 0 1 0 0 1 0 1 1 0 0 0 0 1 0 1 1 1 0 1 1 1 0 1 0 1 1 0 0 1 0 0 1 1 0 1 0 1 1 1 1 1 0 1 1 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 1 1 0 1 1 0 1 0 0 0 1 1 0|
|1 1 0 1 1 0 0 0 1 0 0 0 0 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 0 0 1 1 1 0 1 1 1 0 0 0 1 1 1 1 1 0 1 1 1 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 1 0 0|
|0 1 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 0 1 1 1 0 1 0 1 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 1 0 1 0 1 1 0 1 0 1 0 1 1 0 0 0 0 1 0 0 0 1 1 1 1 1 0 0 1 1 1 0 0 0 0 1 0 0 0 0|
|1 0 0 1 1 0 0 1 1 1 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 0 0 1 1 0 1 0 0 1 0 0 0 0 1 0 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0|
|1 1 0 0 0 1 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 0 0 0 0 1 0 0 1 1 0 0 1 1 1 1 0 1 1 1 1 0 0 1 1 0 1 1 1 0 0 0 0 0 1 0 1 1 0 1 0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 1 0 1 1 0 0 0 0 1 1 0 1 0 1 1 1 0 0 0 0 1 0 0 0 0|
|1 1 0 0 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 0 0 1 0 0 0 1 1 1 1 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 0 0 1 1 1 0 0 0 1 0 0 1|
|1 0 1 1 1 0 0 0 0 0 1 0 1 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 0 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 1 0 1 0 1 1 1 0 0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 0 1 1 1 0 1 1|
|1 0 0 1 0 0 0 1 1 0 1 0 0 1 1 0 0 1 0 0 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0 1 0 0 1 1 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 1 0 0 1 1 1 0 0 0 1 0 0 1 0 1 1 0 1 0 1 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 1 0 0 1 1 1 1 1 0|
|0 1 0 0 0 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 1 0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 1 0 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0 1 0 1 1 0 1 1 1 1 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0|
|0 0 1 0 1 0 0 0 0 0 0 1 1 1 1 0 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 1 1 0 1 1 0 0 0 0 1 1 1 0 0 1 1 0 0 1 1 1 0 1 0 0 1 1 0 0 1 1 1 0 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0|
|1 0 0 0 1 0 1 0 0 1 1 1 1 1 0 1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 1 1 1 0 0 1 0 0 1 1 1 0 1 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 1 0|
|1 0 0 1 1 0 0 0 1 1 0 1 0 0 0 1 1 1 0 1 0 0 1 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 0 0 1 0 1 0 1 1 1 1 0 0 1 0 1 0 0 0 0 1 0 1 1 1 1 0 1 1 1 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 1 1 0 1|
|0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0|
|0 1 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 1 0 1 1 0 1 1 1 0 0 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 1 0 1 0 1 1 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0|
|1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1|
|1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 0 1 1 1 1 0 0 1 0 1 0 1 1 0 0 0 0 1 1 0 0 1 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 0 1|
|0 0 1 0 1 0 0 1 0 1 1 1 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 0 1 0 0 1 1 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 1 1 1 0 1 0 0 0 0 1 0 0|
|0 1 1 0 1 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 1 0 1 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 1 1 1 0|
|0 0 1 0 0 0 1 1 1 1 1 1 1 0 1 1 0 1 1 0 0 1 1 0 0 1 0 0 0 1 0 1 1 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 1 1 1 1 0 1 0 0 1 0 1 0 0 0 0 1 1 0 1 1 1 0 1 1 1 0 0 1 1 0 1 0 1 0 1 1 0 0 0 0|
|1 0 0 1 0 0 1 1 1 1 1 1 1 0 1 1 0 1 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 1 1 1 0 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 1 0 1 1 0 0 1 0 1 0 0 0 0 1 0 1 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1|
|0 0 0 1 1 1 0 1 1 1 0 0 0 1 0 0 0 1 0 0 1 1 1 0 0 0 0 1 0 0 1 0 1 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 1 0 1 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 1 1 0 0|
|0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 1 0 1 1 0 0 1 0 0 1 0 0 1 0 1 1 0 1 0 0 1 1 1 1 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 1 0 0 1 0 0 0 0 0 1 0 1 1 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 0 1 0 0 1 1 1|
|0 0 0 0 1 0 0 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 1 0 1 0 1 1 0 0 1 0 0 1 1 1 1 0 0 0 1 1 0 1 0 0 1 1 0 1 1 0 0 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 0 0 1 1 0 1 0 1 1 0 1 1 0|
|1 0 0 1 1 1 1 0 0 1 1 1 0 1 1 1 0 0 0 0 0 0 0 1 1 0 1 0 1 0 1 1 0 0 1 0 0 0 1 1 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 1 1 0 0 1 0 1 1 1 1 0 0 1 1 0 1 1|
|0 1 0 1 0 0 0 1 0 1 1 1 0 1 0 1 1 0 0 0 1 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 1 0 0 1 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 0 0 1 0 0 1 1 0 0 1|
|0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 1 0 1 1 1 1 1 0 0 1 0 1 1 0 0 1|
|0 0 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 1 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 1 1 1 0 0 1 0 0 0 0 1 0 0 1|
|0 1 1 0 0 1 1 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 1 0 1 1 1 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0 1 1 0 0 1 1 1 0 0 0 0 1 0 0 1 0 0|
|1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 1 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 1|
|0 0 0 0 0 1 0 0 1 1 0 0 0 1 1 0 1 1 1 0 0 0 0 0 0 1 0 0 1 0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 0 1 1 0 0 0 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 1 1 1 0 0 0 1 0 0 1 0|
|0 1 0 1 0 1 1 0 1 0 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 1 0 1 0 1 0 1 1 0 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 0 1 1 0 0 0 1|
|1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 0 0 0 0 0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 1 1 0 0 0 0 0 0 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1 0|
|1 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 1 1 0 0 1 1 0 1 1 0 1 0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 1 1 1 0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 0 1 0 0 1 1 0|
|1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 1 0 1 1 1 1 0 1 0 0 1 0|
|0 0 0 0 0 1 1 1 0 0 1 1 1 1 0 1 0 0 1 0 0 0 1 0 1 0 1 1 0 0 1 1 0 0 1 0 0 1 0 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 0 0 1 1 0 0 0 1 0 1 1 0 1 1 1 0 1 0 1 0 1 0 1 0 0 0 1 0 0 0 1 1 0 1 1 1 1 1 0|
|1 0 0 0 1 1 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 1 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1|
|0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 1 1 0 1 0 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 0 1 0 1 0 0 0 1 0 1 0 0 1 1 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 1 0 0 1 0 0 1 1 1 0 1 1 0 0 1 0|
|0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 1 0 0 0 0 1 1 0|
|0 0 0 0 1 1 0 0 0 1 1 1 1 1 1 0 1 0 0 0 1 0 0 0 1 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 1 1 1 0 0 0 1 1 1 0 1 0 1 0 1 0|
|1 1 1 0 0 1 0 1 0 0 1 0 1 1 0 0 0 1 0 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 1 0 1 1 0 1 1 1 1 1 1 1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 1|
|0 1 0 1 1 0 1 0 0 1 1 1 0 0 1 0 0 0 0 0 1 0 1 1 1 0 1 0 1 0 0 0 1 0 1 1 0 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0|
|0 1 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 1 0 1 0 0 0 0 1 0 1 0 1 0 1 0 0 1 1 1 0 1 0 0 1 0 1 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 0 0 0 1 1 0 1 1 0 1 0 0 0 0 0 1 1 0 1 0|
|1 0 0 1 0 0 0 0 1 1 0 1 1 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 1 1 0 0 1 0 1 1 1 0 0 0 0 1 0 1 1 0 1 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0|
|0 1 0 0 0 1 0 1 1 1 0 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 1 0 0 1 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 1 0 1 1 1 1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 1 0 1|
|0 0 0 1 1 0 0 0 1 1 0 0 1 0 0 1 0 0 1 1 1 0 0 0 0 1 1 0 1 0 0 1 0 1 0 0 1 0 1 1 0 0 0 0 1 1 0 1 1 0 0 0 1 0 1 0 0 0 1 0 1 1 1 0 1 0 0 0 1 1 1 1 1 0 0 1 0 1 1 1 1 1 0 1 1 0 0 1 1 1 0 1 1 0 1 1 0 1 0|
|0 1 0 0 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 1 0 0 1 1 0 1 0 1 1 0 0 1 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 1 1 0 0 0 1 1 0 1|
|1 1 0 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 1 0 0 0 0 0 1 0 1 1 0 1 0 1 0 0 0 1 1 1 1 0 1 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 1 0 0|
|0 1 0 1 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 0 1 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 0 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 1 1 0 0 1 0 0 1 1 1 0 0 1 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 1 0 0|
|0 1 1 1 1 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 1 0 0 1 1 1 0 1 1 0 1 0 1 1 1 0 1 0 0 1 1 0 0 1 0 1 0 0 0 1 1 0 1 0 1 1 0 1 1 0 0 1 1 0 1 0 0 0 1 0 0 1 1 0 1 1 1 0 1 0 0 1 1 1|
|0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 1 0 1 1 1 0 0 0 0 1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0|
|1 1 0 1 0 0 1 0 0 0 0 1 1 1 0 0 1 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 1 0 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 0 1 0 1 0 0 1 1 1 0 0 0 1 0 0 1 1 0 1 1 0 0 0 1 0 1 0 0 0 0 0|
|0 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 0 0 1 0 1 1 1 1 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 1 1 1 0 1 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0|
|1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 1 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 0 0 0 1 1 0 1 1 0 1 1 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 1 1 0 1 1 1 0|
|1 0 0 1 1 1 1 0 0 0 0 0 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0 1 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 0 1 1 0 0 0 1 0 0 1 0|
|1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 0 1 0 0 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 1 1 1 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0|
|0 0 0 0 1 1 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1 0 1 1 0 1 0 0 1 1 1 0 0 1 0 1 0 0 0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 1 0 1 1 1 1 1 1 1 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0|
|0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 1 1 0 1 0 1 1 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 0 1 1|
|0 1 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 0 0 1 0 1 1 0 1 1 1 0 0 0 1 0 1 0 1 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 1 1 1 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 1 1 0 0 0 1 1 0 1 0 0|
|0 1 1 0 1 1 0 0 1 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 1 1 0 0 0 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0 0|
|0 1 1 1 0 0 1 0 1 0 1 0 0 1 1 0 1 0 0 0 1 0 1 1 1 0 1 1 0 1 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 1 1 1 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 0 1 1 0 1 0 1 0 1 1 0 1 1 0 0 0 0 0 1 0|
|0 1 0 1 0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 1 1 1 1 1 0 0 1 0 1 1 1 1 1 0 0|
|0 1 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 1 1 0 1 1 1 0 1 0 1 0 1 0 1 0 0 1 1 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 1 1 1 1 0 1 1 0 0|
|1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 1 1 0 0 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 1 1 0 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 1 1 0|
|1 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 1 0 1 1 1 1 0 1 0 0 0 1 0 1 1 1 0 1 0 0 1 1 0 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0|
|1 0 0 1 1 0 1 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 1 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0|
|0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 1 1 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0|
|1 0 1 0 1 1 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0 1 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 1 1|
|1 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 0 0 1 1 0 1 1 1 0 0 1 1 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 0 1 0 0 1 1 1|
|0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 1 1 0 1 0 1 1 0 0 1 0 1 1 0 0 1 0 1 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 1 0|
|0 0 1 0 0 1 0 0 1 0 0 0 1 1 0 0 1 1 1 0 0 0 1 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 1 1 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0|
|0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1|
|1 1 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 0 0 1 1 1 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 1 1 0 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0|
|0 0 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 1 1 1 0 1 0 0 1 1 0 0 0 1 1 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 1 1 1 1 1 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0|
|0 1 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 1 0 0 1 0 0 1 0 0 0 1 1 0 1 1 1 1 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 1 0 0 1 1 1 1 0 0 0 0 1|
|1 1 1 0 1 1 0 0 0 1 0 1 0 1 1 0 1 1 1 0 0 1 0 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 1 0 0 0 0 0 0 0 1 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 1 1 0|
|0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 0 0 0 0 0 0 1 0 1 0 1 1 0 0 1 1 0 1 0 1 0 1 0 0 1 0 1 0 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 1 0 1 1 0 0 1 1 1 1 0 0 1|
|0 0 0 0 0 1 1 0 0 0 0 1 0 0 1 1 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 0 1|
|0 0 0 1 0 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0 1 0 0 0 0 1 1 0 1 0 1 1 0 1 1 1 0 0 0 0 1 0 1 0 0 0 0 1 1 0 1 1 1 0 0 1 0 0 0 0 1 1 1 0 1 1 1 1 0 1 1 0 0 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1|
|0 0 1 0 1 0 0 0 1 1 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 0 0 1 1 0 1 1 1 1 0 0 1 0 0 0 0 0 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 1 1 1 0|
|1 0 0 0 0 1 0 1 0 1 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1 0 1 1 0 0 1 1 1 0 1 1 1 0 0 1 0 0 0 0 1 1 1 1 1 0 0 1 0 0 1 0 0 0 1 0 1 1 0 0 1|
|1 0 1 1 1 0 0 1 0 0 1 1 0 1 0 0 1 1 1 0 0 1 1 1 0 1 0 0 0 0 0 1 1 1 1 0 1 0 1 0 1 0 0 0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0 1 0 1 0 0 1 0 0 0 0|
|1 0 1 0 0 0 1 1 1 1 0 1 0 1 1 0 0 1 0 0 1 1 1 0 1 1 0 1 1 0 0 1 1 0 1 0 0 0 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 0 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 1 0 0 0 0 1|
|1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1 0 1 1 1 0 1 1 0 1 1 0 1 0 0 0 1 0 1 1 0 1 1 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0|
|0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 1 1 0 1 0 1 0 1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 1 0 1 1 1 0 1 0 1 0 1 1 0|
|0 0 0 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 1 1 1 0 0 0 1 0 0 1 0 1 0 0 1 0 1 1 0 1 1 0 1 0 1 0 1 0 1 0 0 0|
|0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 0 1 0 1 1 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 1 0 1 0 1 0 0 1 0 1 1 0 1 1 1 0 1 0 0 1 0 0 0 1 1 1 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1|
|0 0 0 0 1 1 0 0 1 0 0 1 0 1 0 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 1 0 0 1 1 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 0 1 0 0 0 1 1 0 1|
|0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 1 1 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 1 0 0 0 0 1 1 1 0 0 1 1|
|0 0 1 1 0 0 1 0 1 1 1 0 1 0 0 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 1 1 0 0 0 0 1 1 0 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 1 1 1 1 1 0 0 1 1 0 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0|
|0 1 1 0 1 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 0 1 1 0 1 1 1 0 1 1 0 0 1 0 0 1 1 1 0 1 1 1 1 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1|
|0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 1 1 0 0 0 1 0 1 0 1 1 1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 1 1 1 0 1 0 1 0 0 0 1 1 1 0 1 0|
```

Списки смежности:

```
0 => [1, 4, 6, 10, 12, 13, 14, 15, 16, 19, 20, 23, 24, 28, 32, 37, 40, 41, 42, 44, 48, 51, 55, 59, 61, 62, 63, 71, 72, 73, 75, 76, 80, 83, 88, 89, 90, 91]
1 => [0, 4, 5, 6, 9, 10, 11, 13, 14, 17, 22, 26, 30, 33, 36, 37, 39, 42, 48, 49, 50, 52, 54, 55, 56, 57, 59, 61, 65, 66, 67, 68, 69, 70, 79, 80, 82, 83, 84, 92, 98, 99]
2 => [4, 7, 8, 9, 15, 18, 22, 23, 25, 26, 27, 36, 41, 48, 57, 58, 67, 68, 75, 78, 79, 83, 87, 89, 90, 91, 94, 97, 98]
3 => [5, 6, 9, 10, 11, 12, 15, 16, 20, 28, 29, 32, 33, 35, 39, 49, 51, 53, 56, 57, 59, 60, 61, 62, 63, 68, 69, 73, 77, 79, 84, 86, 89, 93, 96, 97, 99]
4 => [0, 1, 2, 6, 7, 9, 10, 12, 15, 18, 19, 20, 22, 25, 26, 29, 31, 32, 35, 40, 44, 46, 47, 49, 53, 54, 57, 58, 62, 64, 67, 72, 73, 75, 77, 83, 87, 89, 95, 96, 98, 99]
5 => [1, 3, 6, 9, 11, 13, 14, 22, 24, 26, 29, 30, 32, 36, 38, 39, 41, 43, 44, 47, 48, 50, 52, 55, 56, 61, 62, 64, 66, 67, 69, 71, 72, 75, 77, 78, 82, 83, 85, 86, 88, 91, 93, 95]
6 => [0, 1, 3, 4, 5, 7, 8, 14, 19, 21, 27, 28, 32, 34, 35, 36, 37, 39, 41, 43, 46, 49, 54, 56, 58, 59, 60, 62, 64, 66, 68, 70, 71, 73, 75, 76, 77, 79, 80, 81, 84, 85, 86, 90, 97]
7 => [2, 4, 6, 9, 12, 13, 16, 17, 21, 23, 25, 27, 28, 29, 30, 31, 33, 34, 36, 37, 43, 48, 50, 52, 55, 57, 70, 71, 76, 77, 81, 84, 86, 88, 89, 90, 91]
8 => [2, 6, 9, 10, 12, 13, 14, 16, 20, 27, 28, 29, 31, 34, 37, 38, 39, 44, 51, 52, 53, 54, 56, 64, 66, 67, 68, 69, 70, 74, 75, 76, 78, 80, 84, 87, 90, 92, 94, 95, 97, 98, 99]
9 => [1, 2, 3, 4, 5, 7, 8, 12, 14, 17, 19, 20, 25, 27, 28, 29, 31, 32, 33, 35, 37, 38, 41, 44, 45, 47, 49, 50, 51, 52, 53, 55, 56, 60, 64, 66, 67, 76, 81, 83, 86, 87, 88, 90, 91, 93, 97, 98]
10 => [0, 1, 3, 4, 8, 13, 15, 16, 17, 19, 25, 27, 28, 31, 32, 33, 35, 36, 37, 41, 42, 43, 44, 45, 47, 48, 49, 54, 55, 57, 60, 64, 68, 80, 87, 88, 89, 93, 97]
11 => [1, 3, 5, 12, 13, 18, 19, 20, 21, 22, 23, 25, 27, 28, 31, 32, 33, 35, 37, 42, 43, 47, 49, 51, 56, 59, 60, 62, 64, 65, 67, 69, 71, 72, 77, 81, 82, 83, 84, 85, 88, 89, 90, 95]
12 => [0, 3, 4, 7, 8, 9, 11, 15, 18, 19, 25, 27, 37, 39, 42, 43, 44, 45, 46, 47, 48, 50, 51, 53, 54, 56, 59, 60, 62, 65, 70, 72, 73, 77, 78, 86, 92, 93, 97]
13 => [0, 1, 5, 7, 8, 10, 11, 14, 16, 17, 18, 19, 25, 28, 29, 32, 33, 34, 35, 37, 38, 39, 40, 43, 44, 46, 47, 48, 54, 56, 57, 59, 63, 65, 66, 68, 69, 75, 77, 78, 83, 84, 86, 88, 89, 90, 95]
14 => [0, 1, 5, 6, 8, 9, 13, 15, 16, 17, 18, 21, 25, 26, 27, 28, 30, 32, 34, 38, 47, 49, 50, 52, 56, 58, 60, 68, 70, 73, 76, 77, 80, 81, 82, 83, 84, 85, 86, 87, 90, 91, 92, 96, 99]
15 => [0, 2, 3, 4, 10, 12, 14, 17, 19, 20, 27, 31, 32, 33, 36, 39, 40, 41, 42, 43, 51, 53, 55, 56, 57, 62, 63, 65, 70, 71, 73, 79, 80, 84, 85, 86, 87, 91, 94, 95, 96, 98, 99]
16 => [0, 3, 7, 8, 10, 13, 14, 17, 20, 25, 26, 28, 30, 33, 35, 38, 39, 40, 45, 47, 52, 54, 56, 59, 60, 61, 65, 68, 70, 71, 73, 75, 76, 78, 80, 83, 85, 89, 91, 94, 95, 96, 97, 98]
17 => [1, 7, 9, 10, 13, 14, 15, 16, 18, 19, 20, 24, 26, 27, 29, 30, 37, 38, 42, 45, 46, 48, 51, 54, 56, 57, 65, 70, 72, 73, 76, 78, 79, 81, 82, 83, 84, 86, 87, 89, 90, 92]
18 => [2, 4, 11, 12, 13, 14, 17, 19, 21, 25, 27, 31, 38, 41, 43, 46, 52, 53, 55, 56, 61, 62, 63, 66, 67, 70, 71, 72, 74, 77, 78, 81, 82, 83, 85, 87, 88, 89, 93]
19 => [0, 4, 6, 9, 10, 11, 12, 13, 15, 17, 18, 20, 21, 30, 31, 36, 37, 39, 40, 41, 42, 45, 48, 49, 50, 52, 53, 54, 59, 63, 69, 70, 72, 80, 81, 82, 87, 95, 97, 98]
20 => [0, 3, 4, 8, 9, 11, 15, 16, 17, 19, 22, 24, 28, 29, 30, 31, 33, 34, 36, 40, 42, 44, 45, 46, 47, 50, 53, 58, 60, 61, 62, 63, 65, 66, 67, 68, 69, 74, 77, 79, 80, 85, 86, 90, 92, 94, 96, 97, 99]
21 => [6, 7, 11, 14, 18, 19, 26, 27, 29, 37, 49, 52, 54, 56, 61, 66, 70, 79, 80, 81, 83, 85, 89, 90, 91, 92, 93, 94, 95]
22 => [1, 2, 4, 5, 11, 20, 26, 27, 28, 29, 31, 32, 34, 35, 36, 40, 41, 43, 45, 49, 50, 51, 55, 56, 57, 60, 62, 64, 65, 67, 68, 72, 75, 78, 82, 89, 90]
23 => [0, 2, 7, 11, 26, 28, 30, 32, 33, 41, 46, 48, 49, 51, 54, 60, 63, 66, 68, 83, 84, 87, 89, 91, 96, 99]
24 => [0, 5, 17, 20, 28, 31, 35, 36, 38, 43, 45, 47, 55, 56, 58, 59, 60, 61, 64, 66, 68, 69, 74, 75, 78, 79, 81, 82, 90, 94, 95, 96, 97, 99]
25 => [2, 4, 7, 9, 10, 11, 12, 13, 14, 16, 18, 27, 32, 33, 35, 36, 40, 45, 48, 49, 50, 53, 59, 60, 63, 66, 72, 76, 79, 80, 81, 84, 86, 88, 89, 90, 92, 97]
26 => [1, 2, 4, 5, 14, 16, 17, 21, 22, 23, 30, 31, 36, 37, 39, 42, 43, 45, 48, 50, 53, 57, 59, 60, 68, 71, 74, 76, 77, 86, 91, 94, 96, 97, 98]
27 => [2, 6, 7, 8, 9, 10, 11, 12, 14, 15, 17, 18, 21, 22, 25, 29, 31, 32, 33, 38, 43, 44, 48, 49, 50, 56, 60, 61, 62, 63, 65, 68, 70, 75, 76, 78, 79, 80, 82, 83, 84, 87, 88, 90, 92, 94, 95]
28 => [0, 3, 6, 7, 8, 9, 10, 11, 13, 14, 16, 20, 22, 23, 24, 34, 37, 39, 41, 45, 46, 47, 50, 51, 53, 54, 62, 64, 65, 67, 69, 71, 72, 75, 77, 82, 84, 86, 87, 88, 90, 94, 95, 97, 99]
29 => [3, 4, 5, 7, 8, 9, 13, 17, 20, 21, 22, 27, 30, 32, 33, 35, 36, 38, 44, 46, 47, 50, 52, 57, 58, 60, 64, 66, 67, 68, 69, 70, 76, 81, 83, 87, 88, 93, 96, 97]
30 => [1, 5, 7, 14, 16, 17, 19, 20, 23, 26, 29, 31, 32, 34, 37, 38, 39, 40, 43, 46, 47, 48, 57, 58, 59, 62, 63, 64, 65, 68, 73, 75, 76, 77, 81, 85, 86, 87, 88, 92, 94, 97, 98, 99]
31 => [4, 7, 8, 9, 10, 11, 15, 18, 19, 20, 22, 24, 26, 27, 30, 33, 34, 35, 36, 40, 41, 43, 46, 47, 49, 50, 53, 55, 56, 61, 62, 66, 68, 72, 74, 75, 81, 82, 84, 86, 89, 90, 92, 94, 95, 97, 98]
32 => [0, 3, 4, 5, 6, 9, 10, 11, 13, 14, 15, 22, 23, 25, 27, 29, 30, 33, 37, 38, 40, 44, 47, 48, 56, 57, 58, 63, 64, 68, 76, 78, 79, 80, 81, 83, 84, 87, 89, 90, 91, 92, 95, 96, 98, 99]
33 => [1, 3, 7, 9, 10, 11, 13, 15, 16, 20, 23, 25, 27, 29, 31, 32, 37, 46, 49, 51, 53, 55, 58, 60, 63, 64, 66, 69, 71, 73, 75, 78, 79, 80, 83, 85, 86, 87, 89, 92, 95, 96, 99]
34 => [6, 7, 8, 13, 14, 20, 22, 28, 30, 31, 41, 43, 44, 45, 49, 58, 60, 62, 63, 65, 73, 79, 81, 82, 84, 86, 87, 88, 89, 90, 93, 95, 96, 99]
35 => [3, 4, 6, 9, 10, 11, 13, 16, 22, 24, 25, 29, 31, 36, 38, 40, 41, 45, 54, 59, 61, 64, 65, 67, 69, 73, 76, 77, 80, 86, 87, 88, 91, 96, 99]
36 => [1, 2, 5, 6, 7, 10, 15, 19, 20, 22, 24, 25, 26, 29, 31, 35, 42, 45, 46, 50, 52, 53, 54, 57, 59, 63, 66, 68, 69, 70, 71, 72, 76, 77, 83, 84, 87, 88, 89, 94, 97]
37 => [0, 1, 6, 7, 8, 9, 10, 11, 12, 13, 17, 19, 21, 26, 28, 30, 32, 33, 42, 43, 48, 49, 51, 61, 63, 65, 72, 73, 74, 75, 76, 77, 78, 81, 83, 87, 91, 97, 98, 99]
38 => [5, 8, 9, 13, 14, 16, 17, 18, 24, 27, 29, 30, 32, 35, 39, 42, 44, 45, 49, 50, 53, 54, 55, 64, 82, 83, 84, 85, 87, 89, 90, 91, 95, 98]
39 => [1, 3, 5, 6, 8, 12, 13, 15, 16, 19, 26, 28, 30, 38, 45, 51, 52, 53, 57, 58, 59, 60, 64, 65, 66, 70, 72, 74, 76, 77, 82, 85, 87, 88, 91, 92, 94, 95, 99]
40 => [0, 4, 13, 15, 16, 19, 20, 22, 25, 30, 31, 32, 35, 43, 44, 46, 49, 50, 51, 57, 58, 61, 62, 63, 64, 65, 66, 67, 69, 70, 73, 75, 76, 80, 86, 89, 92, 93, 98]
41 => [0, 2, 5, 6, 9, 10, 15, 18, 19, 22, 23, 28, 31, 34, 35, 42, 44, 45, 48, 49, 51, 52, 54, 57, 58, 59, 62, 65, 66, 71, 72, 73, 77, 79, 82, 84, 87, 90, 91, 92, 94, 97, 98]
42 => [0, 1, 10, 11, 12, 15, 17, 19, 20, 26, 36, 37, 38, 41, 43, 48, 49, 50, 51, 61, 67, 68, 72, 75, 77, 78, 82, 85, 86, 88, 90, 91, 92, 93, 95, 98]
43 => [5, 6, 7, 10, 11, 12, 13, 15, 18, 22, 24, 26, 27, 30, 31, 34, 37, 40, 42, 47, 49, 52, 56, 57, 58, 59, 61, 64, 65, 69, 71, 72, 74, 75, 76, 78, 80, 82, 84, 88, 92, 93, 95, 96, 97, 98]
44 => [0, 4, 5, 8, 9, 10, 12, 13, 20, 27, 29, 32, 34, 38, 40, 41, 46, 48, 49, 51, 53, 54, 56, 57, 58, 62, 69, 72, 73, 74, 78, 80, 82, 91, 92, 97, 98, 99]
45 => [9, 10, 12, 16, 17, 19, 20, 22, 24, 25, 26, 28, 34, 35, 36, 38, 39, 41, 47, 49, 50, 53, 55, 59, 61, 64, 65, 68, 74, 76, 77, 81, 83, 84, 87, 90, 91, 92, 94, 95, 98]
46 => [4, 6, 12, 13, 17, 18, 20, 23, 28, 29, 30, 31, 33, 36, 40, 44, 47, 50, 56, 57, 62, 63, 70, 72, 76, 78, 79, 81, 84, 87, 88, 89, 92, 97, 98]
47 => [4, 5, 9, 10, 11, 12, 13, 14, 16, 20, 24, 28, 29, 30, 31, 32, 43, 45, 46, 50, 52, 53, 54, 55, 60, 71, 73, 74, 77, 78, 80, 83, 84, 85, 86, 90, 91, 92, 94, 96, 98]
48 => [0, 1, 2, 5, 7, 10, 12, 13, 17, 19, 23, 25, 26, 27, 30, 32, 37, 41, 42, 44, 51, 53, 54, 56, 57, 58, 59, 60, 61, 65, 68, 70, 76, 77, 79, 82, 86, 89, 93, 95, 99]
49 => [1, 3, 4, 6, 9, 10, 11, 14, 19, 21, 22, 23, 25, 27, 31, 33, 34, 37, 38, 40, 41, 42, 43, 44, 45, 50, 52, 55, 56, 57, 67, 70, 74, 81, 82, 83, 89, 90, 92]
50 => [1, 5, 7, 9, 12, 14, 19, 20, 22, 25, 26, 27, 28, 29, 31, 36, 38, 40, 42, 45, 46, 47, 49, 52, 54, 55, 57, 59, 62, 72, 73, 74, 77, 78, 83, 84, 86, 87, 89, 95, 96, 98]
51 => [0, 3, 8, 9, 11, 12, 15, 17, 22, 23, 28, 33, 37, 39, 40, 41, 42, 44, 48, 55, 58, 60, 62, 63, 64, 67, 69, 70, 71, 76, 78, 79, 81, 82, 83, 84, 86, 90, 91, 96]
52 => [1, 5, 7, 8, 9, 14, 16, 18, 19, 21, 29, 36, 39, 41, 43, 47, 49, 50, 53, 54, 56, 57, 58, 59, 60, 70, 71, 72, 77, 79, 80, 81, 82, 85, 86, 88, 93, 97, 99]
53 => [3, 4, 8, 9, 12, 15, 18, 19, 20, 25, 26, 28, 31, 33, 36, 38, 39, 44, 45, 47, 48, 52, 54, 58, 60, 61, 62, 64, 68, 69, 70, 71, 72, 75, 77, 78, 79, 80, 81, 83, 84, 87, 88, 89, 91, 92, 94, 95, 97, 98]
54 => [1, 4, 6, 8, 10, 12, 13, 16, 17, 19, 21, 23, 28, 35, 36, 38, 41, 44, 47, 48, 50, 52, 53, 56, 59, 61, 64, 66, 68, 72, 74, 81, 84, 87, 90, 91, 92, 96, 97, 99]
55 => [0, 1, 5, 7, 9, 10, 15, 18, 22, 24, 31, 33, 38, 45, 47, 49, 50, 51, 57, 59, 60, 62, 64, 68, 69, 70, 71, 73, 76, 80, 81, 85, 86, 89, 92, 97]
56 => [1, 3, 5, 6, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 21, 22, 24, 27, 31, 32, 43, 44, 46, 48, 49, 52, 54, 57, 61, 67, 68, 69, 72, 75, 76, 77, 78, 81, 82, 87, 90, 91, 94, 97]
57 => [1, 2, 3, 4, 7, 10, 13, 15, 17, 22, 26, 29, 30, 32, 36, 39, 40, 41, 43, 44, 46, 48, 49, 50, 52, 55, 56, 59, 61, 65, 66, 68, 70, 71, 73, 74, 77, 78, 80, 84, 87, 88, 90, 91, 92, 94, 97, 98, 99]
58 => [2, 4, 6, 14, 20, 24, 29, 30, 32, 33, 34, 39, 40, 41, 43, 44, 48, 51, 52, 53, 64, 71, 73, 74, 75, 76, 80, 81, 85, 87, 89, 98]
59 => [0, 1, 3, 6, 11, 12, 13, 16, 19, 24, 25, 26, 30, 35, 36, 39, 41, 43, 45, 48, 50, 52, 54, 55, 57, 61, 65, 66, 67, 70, 72, 75, 76, 77, 81, 84, 85, 87, 88, 92, 94]
60 => [3, 6, 9, 10, 11, 12, 14, 16, 20, 22, 23, 24, 25, 26, 27, 29, 33, 34, 39, 47, 48, 51, 52, 53, 55, 61, 62, 64, 66, 72, 76, 78, 79, 82, 86, 90, 97, 98]
61 => [0, 1, 3, 5, 16, 18, 20, 21, 24, 27, 31, 35, 37, 40, 42, 43, 45, 48, 53, 54, 56, 57, 59, 60, 63, 68, 69, 70, 74, 77, 78, 79, 82, 83, 86, 88, 90, 92, 93, 94, 96, 97, 98]
62 => [0, 3, 4, 5, 6, 11, 12, 15, 18, 20, 22, 27, 28, 30, 31, 34, 40, 41, 44, 46, 50, 51, 53, 55, 60, 66, 67, 71, 74, 80, 81, 86, 87, 88, 90, 91, 95, 98]
63 => [0, 3, 13, 15, 18, 19, 20, 23, 25, 27, 30, 32, 33, 34, 36, 37, 40, 46, 51, 61, 64, 66, 72, 75, 76, 77, 78, 80, 81, 83, 93]
64 => [4, 5, 6, 8, 9, 10, 11, 22, 24, 28, 29, 30, 32, 33, 35, 38, 39, 40, 43, 45, 51, 53, 54, 55, 58, 60, 63, 65, 66, 67, 68, 69, 70, 71, 72, 76, 78, 79, 86, 93, 94, 96, 98]
65 => [1, 11, 12, 13, 15, 16, 17, 20, 22, 27, 28, 30, 34, 35, 37, 39, 40, 41, 43, 45, 48, 57, 59, 64, 66, 70, 71, 72, 76, 80, 82, 84, 86, 88, 90, 92, 93, 96, 98, 99]
66 => [1, 5, 6, 8, 9, 13, 18, 20, 21, 23, 24, 25, 29, 31, 33, 36, 39, 40, 41, 54, 57, 59, 60, 62, 63, 64, 65, 67, 73, 75, 80, 82, 86, 88, 89, 90, 94, 95, 97]
67 => [1, 2, 4, 5, 8, 9, 11, 18, 20, 22, 28, 29, 35, 40, 42, 49, 51, 56, 59, 62, 64, 66, 69, 70, 74, 75, 77, 82, 84, 85, 86, 88, 90, 91, 94, 95, 97]
68 => [1, 2, 3, 6, 8, 10, 13, 14, 16, 20, 22, 23, 24, 26, 27, 29, 30, 31, 32, 36, 42, 45, 48, 53, 54, 55, 56, 57, 61, 64, 69, 70, 74, 79, 82, 83, 85, 87, 89, 90, 92, 93, 98]
69 => [1, 3, 5, 8, 11, 13, 19, 20, 24, 28, 29, 33, 35, 36, 40, 43, 44, 51, 53, 55, 56, 61, 64, 67, 68, 70, 72, 78, 82, 84, 85, 86, 87, 88, 91, 93, 94, 95, 96, 97]
70 => [1, 6, 7, 8, 12, 14, 15, 16, 17, 18, 19, 21, 27, 29, 36, 39, 40, 46, 48, 49, 51, 52, 53, 55, 57, 59, 61, 64, 65, 67, 68, 69, 71, 80, 84, 86, 87, 88, 91, 92, 93, 94, 96, 97]
71 => [0, 5, 6, 7, 11, 15, 16, 18, 26, 28, 33, 36, 41, 43, 47, 51, 52, 53, 55, 57, 58, 62, 64, 65, 70, 72, 76, 77, 79, 80, 81, 88, 89, 94, 97, 98]
72 => [0, 4, 5, 11, 12, 17, 18, 19, 22, 25, 28, 31, 36, 37, 39, 41, 42, 43, 44, 46, 50, 52, 53, 54, 56, 59, 60, 63, 64, 65, 69, 71, 79, 85, 89, 98]
73 => [0, 3, 4, 6, 12, 14, 15, 16, 17, 30, 33, 34, 35, 37, 40, 41, 44, 47, 50, 55, 57, 58, 66, 74, 77, 91, 94, 97, 98]
74 => [8, 18, 20, 24, 26, 31, 37, 39, 43, 44, 45, 47, 49, 50, 54, 57, 58, 61, 62, 67, 68, 73, 75, 76, 77, 80, 81, 86, 87, 88, 89, 93, 95, 97]
75 => [0, 2, 4, 5, 6, 8, 13, 16, 22, 24, 27, 28, 30, 31, 33, 37, 40, 42, 43, 53, 56, 58, 59, 63, 66, 67, 74, 80, 82, 85, 89, 91, 98, 99]
76 => [0, 6, 7, 8, 9, 14, 16, 17, 25, 26, 27, 29, 30, 32, 35, 36, 37, 39, 40, 43, 45, 46, 48, 51, 55, 56, 58, 59, 60, 63, 64, 65, 71, 74, 82, 83, 85, 87, 91, 94, 97, 98, 99]
77 => [3, 4, 5, 6, 7, 11, 12, 13, 14, 18, 20, 26, 28, 30, 35, 36, 37, 39, 41, 42, 45, 47, 48, 50, 52, 53, 56, 57, 59, 61, 63, 67, 71, 73, 74, 81, 82, 86, 92, 93, 97, 98]
78 => [2, 5, 8, 12, 13, 16, 17, 18, 22, 24, 27, 32, 33, 37, 42, 43, 44, 46, 47, 50, 51, 53, 56, 57, 60, 61, 63, 64, 69, 80, 81, 83, 84, 87, 89, 90, 98]
79 => [1, 2, 3, 6, 15, 17, 20, 21, 24, 25, 27, 32, 33, 34, 41, 46, 48, 51, 52, 53, 60, 61, 64, 68, 71, 72, 80, 83, 84, 88, 91, 93, 99]
80 => [0, 1, 6, 8, 10, 14, 15, 16, 19, 20, 21, 25, 27, 32, 33, 35, 40, 43, 44, 47, 52, 53, 55, 57, 58, 62, 63, 65, 66, 70, 71, 74, 75, 78, 79, 81, 85, 86, 88, 91, 94, 95, 96]
81 => [6, 7, 9, 11, 14, 17, 18, 19, 21, 24, 25, 29, 30, 31, 32, 34, 37, 45, 46, 49, 51, 52, 53, 54, 55, 56, 58, 59, 62, 63, 71, 74, 77, 78, 80, 85, 88, 94, 98]
82 => [1, 5, 11, 14, 17, 18, 19, 22, 24, 27, 28, 31, 34, 38, 39, 41, 42, 43, 44, 48, 49, 51, 52, 56, 60, 61, 65, 66, 67, 68, 69, 75, 76, 77, 83, 84, 85, 88, 91, 92, 93, 94, 99]
83 => [0, 1, 2, 4, 5, 9, 11, 13, 14, 16, 17, 18, 21, 23, 27, 29, 32, 33, 36, 37, 38, 45, 47, 49, 50, 51, 53, 61, 63, 68, 76, 78, 79, 82, 85, 86, 88, 96, 97, 98]
84 => [1, 3, 6, 7, 8, 11, 13, 14, 15, 17, 23, 25, 27, 28, 31, 32, 34, 36, 38, 41, 43, 45, 46, 47, 50, 51, 53, 54, 57, 59, 65, 67, 69, 70, 78, 79, 82, 86, 87, 89, 90, 93, 94, 95, 96, 99]
85 => [5, 6, 11, 14, 15, 16, 18, 20, 21, 30, 33, 38, 39, 42, 47, 52, 55, 58, 59, 67, 68, 69, 72, 75, 76, 80, 81, 82, 83, 89, 92, 93, 96, 99]
86 => [3, 5, 6, 7, 9, 12, 13, 14, 15, 17, 20, 25, 26, 28, 30, 31, 33, 34, 35, 40, 42, 47, 48, 50, 51, 52, 55, 60, 61, 62, 64, 65, 66, 67, 69, 70, 74, 77, 80, 83, 84, 88, 90, 95, 99]
87 => [2, 4, 8, 9, 10, 14, 15, 17, 18, 19, 23, 27, 28, 29, 30, 32, 33, 34, 35, 36, 37, 38, 39, 41, 45, 46, 50, 53, 54, 56, 57, 58, 59, 62, 68, 69, 70, 74, 76, 78, 84, 92, 93, 94, 96, 97, 98]
88 => [0, 5, 7, 9, 10, 11, 13, 18, 25, 27, 28, 29, 30, 34, 35, 36, 39, 42, 43, 46, 52, 53, 57, 59, 61, 62, 65, 66, 67, 69, 70, 71, 74, 79, 80, 81, 82, 83, 86, 89, 93, 95, 96, 99]
89 => [0, 2, 3, 4, 7, 10, 11, 13, 16, 17, 18, 21, 22, 23, 25, 31, 32, 33, 34, 36, 38, 40, 46, 48, 49, 50, 53, 55, 58, 66, 68, 71, 72, 74, 75, 78, 84, 85, 88, 90, 92, 95]
90 => [0, 2, 6, 7, 8, 9, 11, 13, 14, 17, 20, 21, 22, 24, 25, 27, 28, 31, 32, 34, 38, 41, 42, 45, 47, 49, 51, 54, 56, 57, 60, 61, 62, 65, 66, 67, 68, 78, 84, 86, 89, 92, 93, 94, 99]
91 => [0, 2, 5, 7, 9, 14, 15, 16, 21, 23, 26, 32, 35, 37, 38, 39, 41, 42, 44, 45, 47, 51, 53, 54, 56, 57, 62, 67, 69, 70, 73, 75, 76, 79, 80, 82, 92]
92 => [1, 8, 12, 14, 17, 20, 21, 25, 27, 30, 31, 32, 33, 39, 40, 41, 42, 43, 44, 45, 46, 47, 49, 53, 54, 55, 57, 59, 61, 65, 68, 70, 77, 82, 85, 87, 89, 90, 91, 93, 95, 97, 98]
93 => [3, 5, 9, 10, 12, 18, 21, 29, 34, 40, 42, 43, 48, 52, 61, 63, 64, 65, 68, 69, 70, 74, 77, 79, 82, 84, 85, 87, 88, 90, 92, 94, 96]
94 => [2, 8, 15, 16, 20, 21, 24, 26, 27, 28, 30, 31, 36, 39, 41, 45, 47, 53, 56, 57, 59, 61, 64, 66, 67, 69, 70, 71, 73, 76, 80, 81, 82, 84, 87, 90, 93, 96, 99]
95 => [4, 5, 8, 11, 13, 15, 16, 19, 21, 24, 27, 28, 31, 32, 33, 34, 38, 39, 42, 43, 45, 48, 50, 53, 62, 66, 67, 69, 74, 80, 84, 86, 88, 89, 92, 96, 97, 99]
96 => [3, 4, 14, 15, 16, 20, 23, 24, 26, 29, 32, 33, 34, 35, 43, 47, 50, 51, 54, 61, 64, 65, 69, 70, 80, 83, 84, 85, 87, 88, 93, 94, 95, 98, 99]
97 => [2, 3, 6, 8, 9, 10, 12, 16, 19, 20, 24, 25, 26, 28, 29, 30, 31, 36, 37, 41, 43, 44, 46, 52, 53, 54, 55, 56, 57, 60, 61, 66, 67, 69, 70, 71, 73, 74, 76, 77, 83, 87, 92, 95]
98 => [1, 2, 4, 8, 9, 15, 16, 19, 26, 30, 31, 32, 37, 38, 40, 41, 42, 43, 44, 45, 46, 47, 50, 53, 57, 58, 60, 61, 62, 64, 65, 68, 71, 72, 73, 75, 76, 77, 78, 81, 83, 87, 92, 96, 99]
99 => [1, 3, 4, 8, 14, 15, 20, 23, 24, 28, 30, 32, 33, 34, 35, 37, 39, 44, 48, 52, 54, 57, 65, 75, 76, 79, 82, 84, 85, 86, 88, 90, 94, 95, 96, 98]
```

Визуализация:



II. Использование DFS для нахождения связных компонент графа и BFS для нахождения кратчайшего пути между случайными двумя вершинами.

Для нахождения связных компонент достаточно вести учет посещаемости вершин (посещена/не посещена) и из еще не посещенных запускать обход в глубину. Сколько раз обход в глубину запускался, столько и компонент связности. (Компонента связности представляет собой список всех включенных в подграф вершин).

Для нахождения кратчайшего расстояния между случайными двумя вершинами из одной вершины source запускается обход в ширину, как только очередной сосед оказывается target вершиной, путь найден. Чтобы была возможность восстановить путь, во время обхода для каждой вершины записывается ее предшественник (список чисел, где индекс – вершина-предшественник, значение – вершина, куда ведет ребро). Если обход закончился, а target не найдена, то логируется, что данные на вход вершины не связны. Так как в худшем случае просматриваются все вершины и все ребра, сложность O(nVertices + nEdges).

Вывод консоли:

```
11:39:50.804 [main] INFO algorithms2019.Main - generated adjacency matrix is written to adjMatrix.txt

11:39:50.873 [main] INFO algorithms2019.Main - converted adjacency matrix into adjacency list is written to adjList.txt

# of components found in graph: 1

[0, 59, 70, 36, 83, 23, 46, 57, 41, 34, 22, 43, 10, 33, 51, 55, 89, 11, 67, 75, 91, 53, 20, 92, 12, 9, 52, 99, 82, 94, 87, 74, 18, 31, 4, 32, 79, 25, 84, 7, 50, 1, 26, 16, 61, 56, 3, 35, 80, 85, 14, 30, 97, 44, 13, 37, 48, 58, 71, 47, 90, 42, 38, 54, 96, 69, 68, 24, 60, 78, 98, 76, 64, 72, 39, 15, 95, 19, 6, 27, 17, 65, 45, 81, 21, 66, 5, 62, 88, 93, 63, 77, 86, 40, 73, 49, 28, 8, 2, 29]

Shortest path found with bfs from 9 to 0 is [9 => 1 => 0]
```

Код (без конфигурационных подробностей)**:**

Комментарий к классу: граф определяется количеством вершин и ребер. Для экземпляра генерируется случайная матрица смежности (вершин по условию 100 -> в матрице 10000 ячеек -> тк граф не ориентирован и без петель доступны, для заполнения доступны 4950 ячеек -> чтобы случайно разместить в этих ячейках 2000 элементов, генерируется случайная перестановка $\binom{4950}{2000}$ – сложность O(n)).

```java
@Getter
@Slf4j
public class UndirectedGraph {
    private int nVertices;
    private int nEdges;
    @Setter
    private Map<Integer, List<Integer>> adjList;

    public UndirectedGraph(int nVertices, int nEdges) {
        this.nVertices = nVertices;
        this.nEdges = nEdges;
    }

    private static List<Integer> apply(List<Integer> list) {
        int index = 0;
        List<Integer> indices = new ArrayList<>();
        for (Integer element : list) {
            if (element == 1) {
                indices.add(index);
            }
            index++;
        }
        return indices;
    }

    /*
    as graph in task is undirected, its adjacency matrix is symmetric
    and as it doesn't have any loops, its matrix has zeros on the main diagonal
     */
    public Integer[][] genRandomAdjMatrix() {
        Integer[][] adjMatrix = new Integer[nVertices][nVertices];
        List<Integer> allCells = new ArrayList<>(Collections.nCopies(nVertices *
nVertices / 2 - (nVertices / 2), 0));
        for (int i = 0; i < nEdges; i++) {
            allCells.set(i, 1);
        }
        Random rnd = new Random(); // generate a random permutation of 2000 elements in
4950
        for (int i = allCells.size() - 1; i > 0; i--) {
            Collections.swap(allCells, i, rnd.nextInt(i));
        }
        int currentIndex = 0;      // fill the adj adjMatrix with random permutation
        outer:
        for (int i = 0; i < nVertices - 1; i++) {
            adjMatrix[i][i] = 0;
            for (int j = i + 1; j < nVertices; j++) {
                if (i != j) {
                    adjMatrix[i][j] = allCells.get(currentIndex);
                    adjMatrix[j][i] = allCells.get(currentIndex);
                    currentIndex++;
                }
                if (currentIndex == allCells.size()) {
                    break outer;
                }
            }
        }
        adjMatrix[nVertices - 1][nVertices - 1] = 0;
        return adjMatrix;
    }

    public Map<Integer, List<Integer>> convertMatrixToLists(Integer[][] adjMatrix) {
        List<List<Integer>> adjMatrixList = new ArrayList<>();
        for (Integer[] matrix : adjMatrix) {
            adjMatrixList.add(Arrays.asList(matrix));
        }
        PrimitiveIterator.OfInt iterator = IntStream.range(0,
adjMatrixList.size()).iterator();
        return adjMatrixList.stream()
                .map(UndirectedGraph::apply)
                .collect(Collectors.toMap(obj -> iterator.next(), Function.identity()));
    }

    public HashSet<Edge> getAllEdges(Map<Integer, List<Integer>> adjList) {
        List<Edge> edges = new ArrayList<>();
        adjList.forEach((from, tos) -> tos.forEach(to -> edges.add(new Edge(from,
to))));
        return new HashSet<>(edges);
    }

    public List<List<Integer>> findComponents() {
        List<List<Integer>> listOfComponents = new ArrayList<>();
        List<Integer> component;
        List<Boolean> visited = new
ArrayList<>(Collections.nCopies(adjList.keySet().size(), false));
        for (Integer vertex : adjList.keySet()) {
            if (!visited.get(vertex)) {
                component = dfs(vertex, visited);
                listOfComponents.add(component);
            }
        }
        return listOfComponents;
    }

    public Optional<List<Integer>> findShortestPath(Integer source, Integer target) {
        List<Boolean> visited = new
ArrayList<>(Collections.nCopies(adjList.keySet().size(), false));
        List<Integer> predecessors = new
ArrayList<>(Collections.nCopies(adjList.keySet().size(), -1));
        if (bfs(source, target, visited, predecessors)) {
            return Optional.of(unrollPath(predecessors, source, target));
        } else {
            Log.debug(String.format("SORRY! Given vertices (%d and %d) are not
connected", source, target));
            return Optional.empty();
        }
    }

    private List<Integer> unrollPath(List<Integer> predecessors, Integer source, Integer
target) {
        List<Integer> path = new ArrayList<>();
        Integer current = target;
        path.add(current);
        while (!current.equals(source)) {
            path.add(predecessors.get(current));
            current = predecessors.get(current);
        }
        Collections.reverse(path);
        return path;
    }

    private boolean bfs(Integer source, Integer target, List<Boolean> visited,
List<Integer> predecessors) {
        LinkedList<Integer> queue = new LinkedList<>();
        queue.offer(source);
        visited.set(source, true);
        while (!queue.isEmpty()) {
            Integer current = queue.pollFirst();
            List<Integer> neighbours = adjList.get(current);
            for (Integer neighbour : neighbours) {
                if (!visited.get(neighbour)) {
                    predecessors.set(neighbour, current);
                    queue.offer(neighbour);
                    visited.set(neighbour, true);
                    if (neighbour.equals(target)) {
                        return true;
                    }
                }
            }
        }
        return false;
    }

    private List<Integer> dfs(Integer startVertex, List<Boolean> visited) {
        Random rnd = new Random();
        List<Integer> verticesInComponent = new ArrayList<>();
        LinkedList<Integer> stack = new LinkedList<>();
        stack.push(startVertex);
        verticesInComponent.add(startVertex);
        Integer topVertex;
        while (!stack.isEmpty()) {
            topVertex = stack.peek();
            visited.set(topVertex, true);
            List<Integer> unvisitedNeighbours = getUnvisitedNeighbours(topVertex,
visited);
            if (unvisitedNeighbours.isEmpty()) {
                stack.pop();
            } else {
                Integer nextVertex =
unvisitedNeighbours.get(rnd.nextInt(unvisitedNeighbours.size()));
                stack.push(nextVertex);
                verticesInComponent.add(nextVertex);
            }
        }
        return verticesInComponent;
    }

    private List<Integer> getUnvisitedNeighbours(Integer vertex, List<Boolean> visited)
{
        return adjList.get(vertex).stream()
                .filter(index -> !visited.get(index))
                .collect(Collectors.toList());
    }
}
```

Вспомогательный класс для описания ребра графа (для визуализации графа нужен список ребер). В классе переопределяется правило сравнения двух ребер таким образом, что ребра, содержащие одинаковые вершины, считаются равными (тк граф неориентированный). Это правило будет использоваться при составлении множества уникальных ребер.

```
@AllArgsConstructor
@Getter
public class Edge {
    private Integer from;
    private Integer to;

    @Override
    public boolean equals(Object o){
        if (this == o) return true;
        if (o instanceof Edge) {
            Edge edge = (Edge) o;
            return from != null && to != null ? (from.equals(edge.from)
                    && to.equals(edge.to)) || (from.equals(edge.to)
                    && to.equals(edge.from)) : to == null && from == null;
        }
        return false;
    }

    @Override
    public int hashCode(){
        return from.hashCode() + to.hashCode();
    }

}
```

В главном классе создается экземпляр класса графа, и на нем вызываются все алгоритмы. Помимо, определяются методы для записи в файл матрицы смежности и списков смежности (тк на консоле плохо читаемо).

```
@Slf4j
public class Main {

    @SneakyThrows
    public static void displayAdjMatrix(Integer[][] randomAdjMatrix) {
        File file = new File("./src/main/resources/adjMatrix.txt");
        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(file));
        for (Integer[] adjMatrix : randomAdjMatrix) {
            bufferedWriter.write(("|"));
            for (int j = 0; j < randomAdjMatrix.length; j++) {
                bufferedWriter.write(adjMatrix[j].toString().concat(" "));
            }
            bufferedWriter.write("|\n");
        }
        bufferedWriter.close();
    }

    @SneakyThrows
    public static void displayAdjList(Map<Integer, List<Integer>> adjLists) {
        FileWriter writer = new FileWriter(new
File("./src/main/resources/adjList.txt"));
        BufferedWriter bufferedWriter = new BufferedWriter(writer);
        adjLists.forEach((k, v) -> {
            try {
                bufferedWriter.write(k.toString().concat(" => "));
                bufferedWriter.write(String.format("%s\n", String.join(",",
v.toString())));
            } catch (IOException e) {
                e.printStackTrace();
            }
        });
        bufferedWriter.close();
    }

    public static void main(String[] args) {
        UndirectedGraph undirectedGraph = new UndirectedGraph(100, 2000);
        Integer[][] randomAdjMatrix = undirectedGraph.genRandomAdjMatrix();
        displayAdjMatrix(randomAdjMatrix);
        log.info("generated adjacency matrix is written to adjMatrix.txt");
        Map<Integer, List<Integer>> adjList =
undirectedGraph.convertMatrixToLists(randomAdjMatrix);
        undirectedGraph.setAdjList(adjList);
        displayAdjList(adjList);
        log.info("converted adjacency matrix into adjacency list is written to
adjList.txt");
        List<List<Integer>> components = undirectedGraph.findComponents();
        System.out.println("# of components found in graph: " + components.size());
        components.forEach(System.out::println);
        Random rnd = new Random();
        Integer randVertex1 = rnd.nextInt(adjList.size());
        Integer randVertex2 = rnd.nextInt(adjList.size());
        Optional<List<Integer>> shortestPath =
undirectedGraph.findShortestPath(randVertex1, randVertex2);
        shortestPath.ifPresent(path -> System.out.printf("Shortest path found with bfs
from %d to %d is %s",
                randVertex1, randVertex2, String.join(" =>",
path.toString().split(","))));
        Visualiser.drawGraph(adjList, undirectedGraph.getAllEdges(adjList));
    }
}
```

Для визуализации использовалась библиотека GraphStream:

```
public class Visualiser {
    public static void drawGraph(Map<Integer, List<Integer>> adjLists, HashSet<Edge>
allEdges) {
        Graph graph = new MultiGraph("Visualiser");
        System.setProperty("org.graphstream.ui.renderer",
"org.graphstream.ui.j2dviewer.J2DGraphRenderer");
        graph.addAttribute("ui.stylesheet", "url('file:style.css')");
        adjLists.keySet().forEach(vertex -> graph.addNode(vertex.toString()));
        PrimitiveIterator.OfInt iterator = IntStream.range(0,
allEdges.size()).iterator();
        allEdges.forEach(edge -> graph.addEdge(Integer.toString(iterator.next()),
edge.getFrom(), edge.getTo(), false));
        graph.forEach(node -> node.addAttribute("ui.label", node.getId()));
        graph.display();
    }
}
```

Файл (.css) с настройками параметров визуализации:

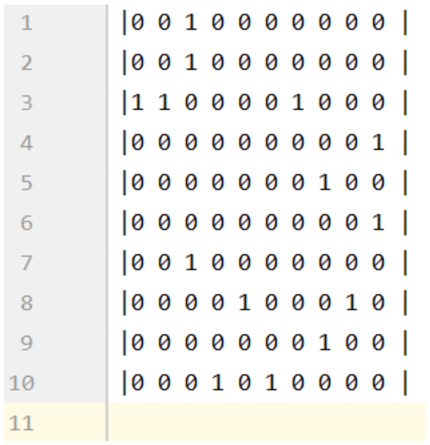
```
graph {
   fill-color: #FDFCF3;
}

node {
   size: 10px, 10px;
   shape: circle;
   fill-color: #4682B4;
   stroke-mode: plain;
   stroke-color: black;
        text-alignment: above;
        text-size: 10;
        text-style: bold-italic;
}

node:clicked {
   fill-color: red;
}
```

```
edge {
        shape: polyline;
        fill-color: #008080;
}
```

ТЕСТ

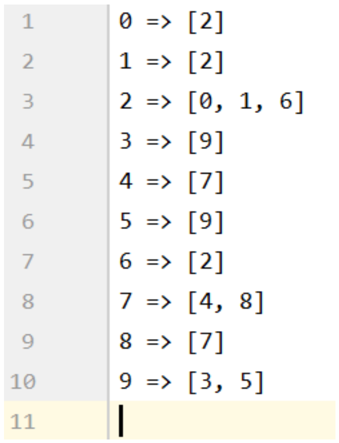
Проверка наглядности работы алгоритмов на небольшом графе с 10 вершинами и 7 ребрами.

Мастрица смежности:



```
|0 0 1 0 0 0 0 0 0 0 |
|0 0 1 0 0 0 0 0 0 0 |
|1 1 0 0 0 0 1 0 0 0 |
|0 0 0 0 0 0 0 0 0 1 |
|0 0 0 0 0 0 0 1 0 0 |
|0 0 0 0 0 0 0 0 0 1 |
|0 0 1 0 0 0 0 0 0 0 |
|0 0 0 0 1 0 0 0 1 0 |
|0 0 0 0 0 0 0 1 0 0 |
|0 0 0 1 0 1 0 0 0 0 |
```

Списки смежности:



```
0 => [2]
1 => [2]
2 => [0, 1, 6]
3 => [9]
4 => [7]
5 => [9]
6 => [2]
7 => [4, 8]
8 => [7]
9 => [3, 5]
```

Вывод консоли:

11:34:11.880 [main] INFO algorithms2019.Main - generated adjacency matrix is written to adjMatrix.txt

11:34:11.945 [main] INFO algorithms2019.Main - converted adjacency matrix into adjacency list is written to adjList.txt

# of components found in graph: 3

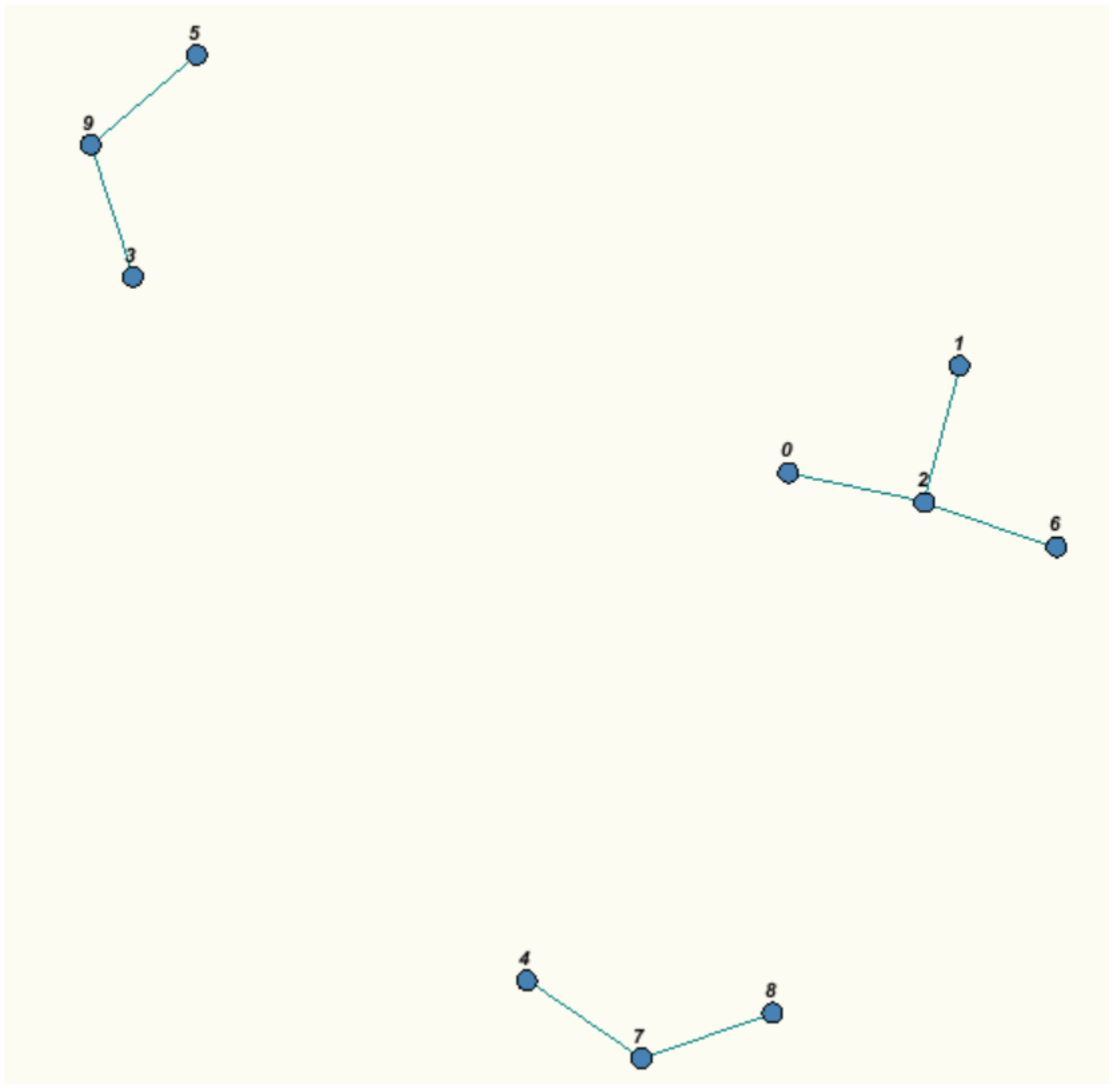
[0, 2, 6, 1]

[3, 9, 5]

[4, 7, 8]

Shortest path found with bfs from 5 to 9 is [5 => 9]

Визуализация:



Вывод: на примере основного с 100 ребрами и 2000 вершинами и тестового с 10 вершинами и 7 ребрами видно, какое представление для каких графов лучше. Если граф сильно связный, как основной, то матричное представление себя оправдывает*: пустых ячеек не так много, и доступ к каждой за O(1)(тк это массив), тогда как на разряженном, как тестовый, видно, что хранение n*n матрицы избыточно: связей мало, эффективнее использовать списки смежности (хотя и доступ к конкретному соседу теперь будет O(n) (тк перебираются все соседи, которых в худшем случае n)). Способ представления списками гораздо удобнее для bfs, где по текущей вершине нужно получить список всех соседей – это O(1), а если бы была матрица, пришлось бы перебирать все n элементов в строке, чтобы отобрать соседей (O(n)). Но при этом, если матричное представление себя по памяти оправдывает, в dfs оно не проигрывает спискам, тк там не нужен список всех соседей для обхода, достаточного одного случайного, а это доступ за O(1).

* отчасти оправдывает, это просто не так плохо, как на разряженных, действительно оправдывает (не хуже, чем списки), если граф полный (сейчас он не полный)/ориентированный с петлями, тогда

сложность по памяти из O(V +E) вырождается в O(V + E^2)~ O(E^2). В таком случае, если в алгоритме нужно проверять существование ребра между вершинами, матричный способ лучше (O(1) против O(n)).