Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

по дисциплине: «Общесистемное программное обеспечение параллельных вычислительных систем»

Студент	Гольцов Илья Сергеевич
Группа	РК6-32М
Тип задания	лабораторная работа №4
Тема лабораторной работы	Волновой алгоритм Ли

Студент _____ **Гольцов И. С.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Грошев С. В.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2022 г.

Оглавление

Задание на лабораторную работу

Написать программу иллюстрирующую работу волнового алгоритма Ли для распределенной сети, представленной в виде графа.

Выполнение лабораторной работы

Алгоритм волновой трассировки (волновой алгоритм, алгоритм Ли) — алгоритм поиска кратчайшего пути на планарном графе. Принадлежит к алгоритмам, основанным на методах поиска в ширину. Алгоритм предназначен для поиска кратчайшего пути от стартовой вершины к конечной, если это возможно. Если невозможно, то алгоритм выдает сообщение о непроходимости.

Работа алгоритма включает в себя три этапа: инициализацию, распространение волны и восстановление пути.

При работе алгоритма на ориентированном графе на этапе инициализации требуется пометить атрибут стартовой вершины начальным значением, а также инициализировать конечную вершину.

Далее, от стартовой вершины порождается шаг в ее потомков, которые еще не были помечены. В атрибут этих вершин записывается число, равное количеству шагов от стартовой вершины (например, на первом шаге это будет 1). Каждая вершина, помеченная числом шагов от стартовой вершины, становится стартовой и из нее порождаются последующие шаги в соседние вершины. Очевидно, что при таком переборе будет найден путь от начальной вершины к конечной, либо очередной шаг из любой порождённой в пути вершины будет невозможен.

Восстановление кратчайшего пути происходит в обратном направлении: при выборе вершины от финишной вершины к стартовой на каждом шаге выбирается вершина, имеющая атрибут расстояния от стартовой на единицу меньше текущей вершины. Очевидно, что таким образом находится кратчайший путь между парой заданных вершин.

Ниже представлен псевдокод волнового алгоритма Ли:

Инициализация

```
Пометить стартовую ячейку
d := 0
```

Распространение волны

```
ЦИКЛ
  ДЛЯ каждой ячейки loc, помеченной числом d
    пометить все соседние свободные непомеченные ячейки числом d + 1
  КЦ
  d := d + 1
ПОКА (финишная ячейка не помечена) И (есть возможность распространения волны)
```

Восстановление пути

```
ЕСЛИ финишная ячейка помечена
ТО
  перейти в финишную ячейку
  ЦИКЛ
    выбрать среди соседних ячейку, помеченную числом на 1 меньше числа в текущей ячейке
    перейти в выбранную ячейку и добавить её к пути
  ПОКА текущая ячейка – не стартовая
  ВОЗВРАТ путь найден
ИНАЧЕ
  ВОЗВРАТ путь не найден
```

Рисунок 1 - Псевдокод волнового алгоритма Ли

В рамках выполнения лабораторной работы была разработана программа, иллюстрирующая работу волнового алгоритма Ли. Решение производилось на графе, представленном на рисунке 2.

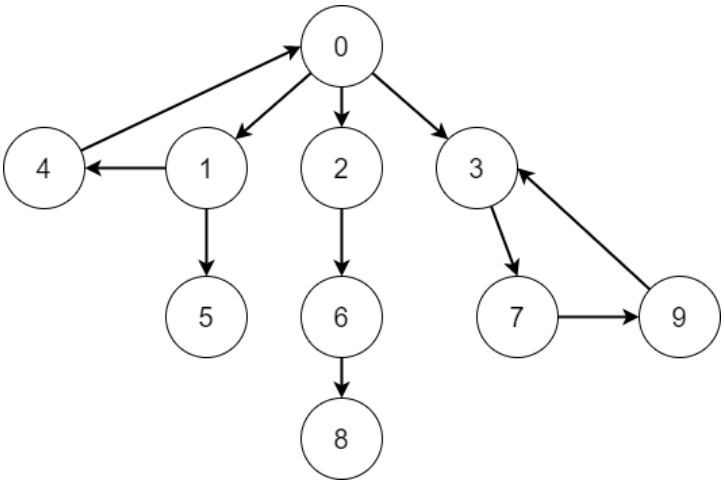

Рисунок 2 – Граф, используемый для решения

Граф задается в программе парами чисел, представляющих его ребра. В начале работы программы происходит считывание структуры графа. На основе заданной структуры строится матрица смежности.

{Adjacency Matrix}

N	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
(0)	0	1	1	1	0	0	0	0	0	0
(1)	0	0	0	0	1	1	0	0	0	0
(2)	0	0	0	0	0	0	1	0	0	0
(3)	0	0	0	0	0	0	0	1	0	0
(4)	1	0	0	0	0	0	0	0	0	0
(5)	0	0	0	0	0	0	0	0	0	0
(6)	0	0	0	0	0	0	0	0	1	0
(7)	0	0	0	0	0	0	0	0	0	1
(8)	0	0	0	0	0	0	0	0	0	0
(9)	0	0	0	1	0	0	0	0	0	0

Рисунок 3 – Полученная программой матрица смежности

Вершиной-инициатором выбирается корневая. Конечной вершиной для заданного примера решения является вершина с индексом 9. Программа разбита на 3 этапа:

- инициализация – в структуру, представляющую собой граф, заносится информация о стартовой вершине, также проставляется маркировка стартовой вершины значением 0; заносится информация о конечной вершине, до которой осуществляется поиск.

- распространение волны – в цикле пока конечная вершина не получит маркировку, отличную от изначально заданной, производится проход по всем посещенным вершинам, где выбираются их потомки и для каждого из еще не посещенных потомков проставляется маркировка на единицу большая, чем у родительской. При этом текущее значение маркировки увеличивается на единицу каждую итерацию после прохода по всем вершинам.

- восстановление пути – в цикле, начиная от конечной вершины, пока не будет достигнута стартовая, изучаются все родители текущей рассматриваемой вершины. Если у родителя маркировка на единицу меньше текущей изучаемой вершины, то выбирается путь через него, соответствующая вершина заносится в структуру данных для хранения пути.

Далее работа алгоритма завершается.

На рисунке 4 представлена иллюстрация работы алгоритма для графа, используемого для решения.

```
Initiator for graph: 0; end: 9
start wave is 0
spread 1 in 1
spread 1 in 2
spread 1 in 3
spread 2 in 4
spread 2 in 5
spread 2 in 6
spread 2 in 7
spread 3 in 8
spread 3 in 9
Path from 0 to 9 is: 0 -> 3 -> 7 -> 9
Process finished with exit code 0
```

Рисунок 4 – Результат работы разработанного алгоритма

На рисунках 5-7 представлена иллюстрация работы алгоритма в виде последовательных состояний сети. Рядом с каждой вершиной обозначена текущая маркировка. Также условно точками изображен текущий фронт волны.

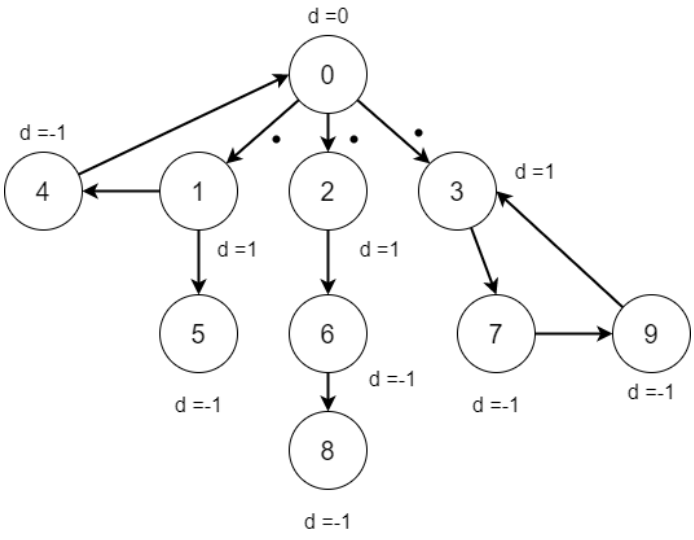

Рисунок 5 – Ход работы алгоритма (шаг 1)

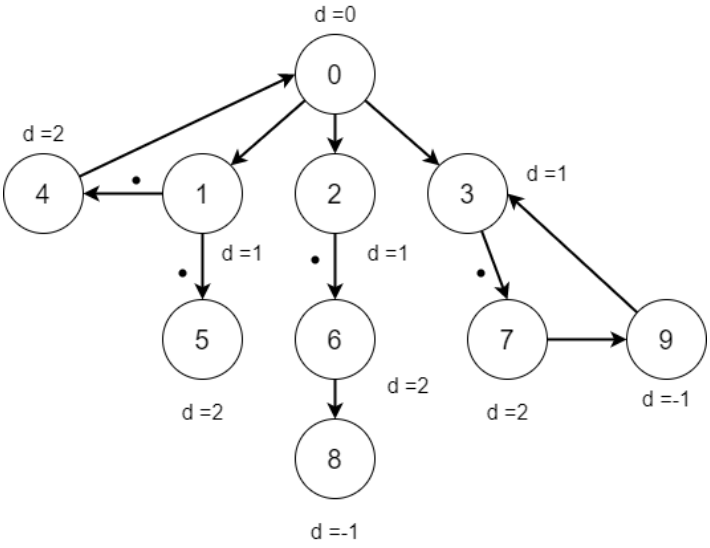

Рисунок 6 – Ход работы алгоритма (шаги 2)

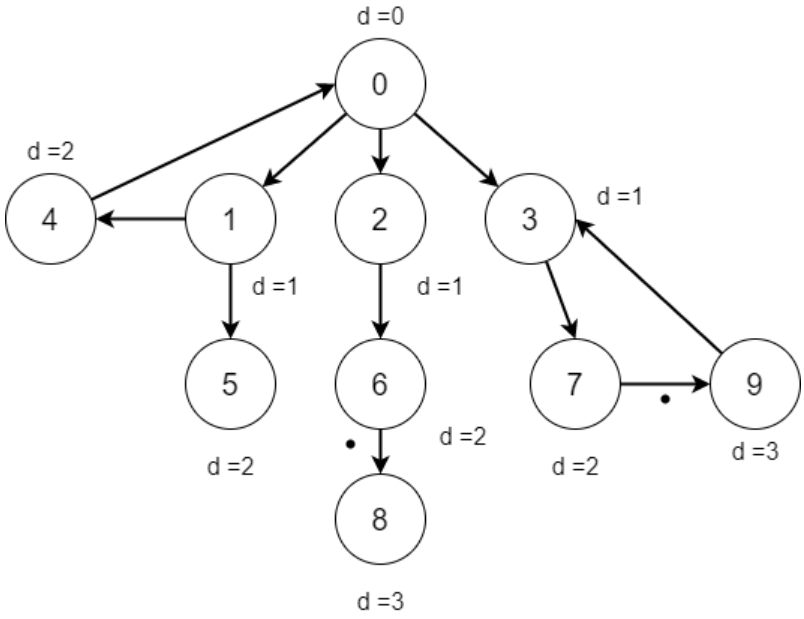

Рисунок 7 – Ход работы алгоритма (шаг 3)

Заключение

В ходе выполнения лабораторной работы был изучен волновой алгоритм Ли и реализован программный код на языке Java, имитирующий работу распределенной сети на заданном графе, в ходе которой при помощи исследуемого алгоритма происходит рассылка сообщений всем вершинам графа, а также нахождение пути от стартовой вершины до заданной конечной.

Код программы

Ниже представлен программный код, отвечающий за выполнение алгоритма:

Листинг 1 – Функция, отвечающая за выполнение алгоритма

```
public void executeLeeWave(Graph g, Integer start, Integer end) {
    // init
    System.out.println("Initiator for graph: " + start + "; end: " + end);
    ((LeeGraph) g).setInitiator(start);

    GraphNode[] nodes = g.getGraphNodes();
    int currentDim = 0;
    ((LeeGraphNode) nodes[start]).setMarkVal(currentDim);
    System.out.println("start wave is " + currentDim);

    // mark
    while (((LeeGraphNode) nodes[end]).getMarkVal() == -1) {
        Arrays.stream(nodes).forEach(n -> {
            var curN = (LeeGraphNode) n;
            if (curN.getMarkVal() != -1) {
                ((LeeGraphNode) n).getSuccessors().forEach(suc -> {
                    LeeGraphNode leeSuc = (LeeGraphNode) suc;
                    if (leeSuc.getMarkVal() == -1) {
                        int val = curN.getMarkVal() + 1;
                        System.out.println("spread " + val + " in " +
suc.getValue());
                        leeSuc.setMarkVal(val);
                    }
                });
            }
        });
        currentDim++;
    }

    // trace
    LeeGraphNode curNode = (LeeGraphNode) nodes[end];
    List<GraphNode> path = new LinkedList<>();
    path.add(curNode);
    while (curNode != nodes[start]) {
        LeeGraphNode finalCurNode = curNode;
        curNode = (LeeGraphNode) curNode.getPredecessors().stream()
                .filter(n -> ((LeeGraphNode) n).getMarkVal() ==
finalCurNode.getMarkVal() - 1)
                .findFirst().orElseThrow(() -> new RuntimeException("path not
found"));
        path.add(curNode);
    }

    System.out.print("Path from " + start + " to " + end + " is: ");
    for (int i = path.size() - 1; i > -1; i--) {
        System.out.print(path.get(i).getValue());
        if (i != 0) {
            System.out.print(" -> ");
        }
    }
}
```