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 №2
Тема лабораторной работы	Волновой алгоритм Финна

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 3
Выполнение лабораторной работы 3
Заключение 7
Код программы 8

Задание на лабораторную работу

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

Выполнение лабораторной работы

Алгоритм Финна – волновой алгоритм, который можно использовать в ориентированных сетях произвольной топологии. Он не требует того, чтобы диаметр сети был известен заранее, но подразумевает наличие уникальных идентификаторов процессов. В сообщениях передаются множества идентификаторов процессов, что приводит к высокой битовой сложности алгоритма.

Процесс S содержит два множества идентификаторов процессов:

- $Inc(S)$ – это множество процессов U таких, что событие в U предшествует последнему произошедшему событию в S;
- $NInc(S)$ – множество процессов U таких, что для всех соседей R процесса U событие в R предшествует последнему произошедшему событию в S. Эта зависимость поддерживается следующим образом.

1. Изначально $Inc(S) = \{S\}$, а $NInc(S) = \emptyset$.
2. Каждый раз, когда одно из множеств пополняется, процесс S посылает сообщение, включая в него $Inc(S)$ и $NInc(S)$.
3. Когда S получает сообщение, включающее множества $Inc(S)$ и $NInc(S)$, полученные идентификаторы включаются в версии этих множеств в процессе S.
4. Когда S получит сообщения от всех соседей по входу, S включается в $NInc(S)$.
5. Когда два множества становятся равны, S выполняет процедуру $return(OK)$.

Из неформального смысла двух множеств следует, что для каждого процесса U такого, что событие в U предшествует некоторому событию E,

выполняется следующее: для каждого соседа R процесса U событие в R также предшествует событию E.

Ниже представлен псевдокод волнового алгоритма Финна:

```
var Inc_p     : мн–во процессов        init {p} ;
    NInc_p    : мн–во процессов        init ∅ ;
    rec_p[q]  : bool for q ∈ In_p      init false ;
                (* индикаторы получения процессом p сообщения от q *
begin if p is initiator then
        forall r ∈ Out_p do send ⟨sets, Inc_p, NInc_p⟩ to r ;
      while Inc_p ≠ NInc_p do
            begin receive ⟨sets, Inc, NInc⟩ from q_0 ;
                  Inc_p := Inc_p ∪ Inc ; NInc_p := NInc_p ∪ NInc ;
                  rec_p[q_0] := true ;
                  if ∀q ∈ In_p : rec_p[q] then NInc_p := NInc_p ∪ {p} ;
                  if Inc_p or NInc_p has changed then
                    forall r ∈ Out_p do send ⟨sets, Inc_p, NInc_p⟩ to r
            end;
      decide
end
```

Рисунок 1 - Псевдокод волнового алгоритма Финна

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

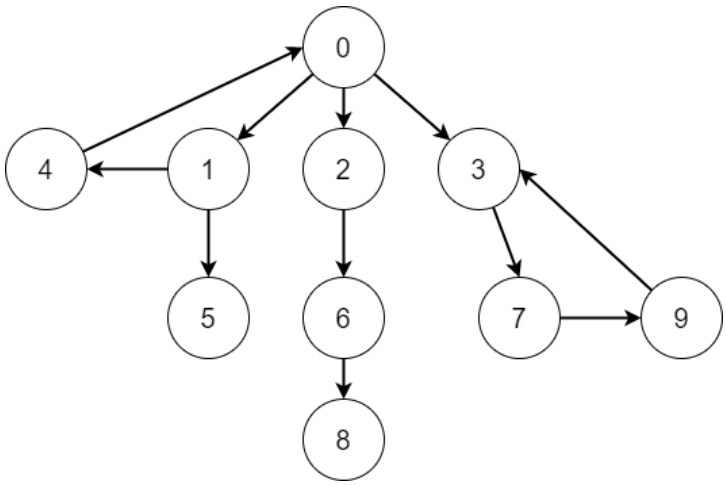

Рисунок 2 – Граф, используемый для решения

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. Затем происходит рассылка сообщений всем потомкам инициатора. Потомки, получая сообщения, обновляют свои множества Inc и $NInc$, далее рассылая сообщения другим вершинам. При этом происходит последовательный перебор всех вершин, что имитирует общий канал передачи данных между вершинами распределенной сети. Каждая вершина в один момент времени производит некоторую операцию, в зависимости от своего состояния. То есть либо ожидает, либо выполняет рассылку сообщений, либо получает сообщения. После того, как для некоторой вершины множества Inc и $NInc$ становятся одинаковыми, происходит принятие решения и алгоритм завершает работу.

```
Initiator for graph: 0
Node 0 will perform the protocol
[[0], [1], [2], [3], [4], [5], [6], [7], [8], [9]]
[[], [], [], [], [], [], [], [], [], []]
Node 1 will perform the protocol
[[0], [0, 1], [2], [3], [4], [5], [6], [7], [8], [9]]
[[], [1], [], [], [], [], [], [], [], []]
Node 2 will perform the protocol
[[0], [0, 1], [0, 2], [3], [4], [5], [6], [7], [8], [9]]
[[], [1], [2], [], [], [], [], [], [], []]
Node 3 will perform the protocol
[[0], [0, 1], [0, 2], [0, 3], [4], [5], [6], [7], [8], [9]]
[[], [1], [2], [], [], [], [], [], [], []]
Node 4 will perform the protocol
[[0], [0, 1], [0, 2], [0, 3], [0, 1, 4], [5], [6], [7], [8], [9]]
[[], [1], [2], [], [1, 4], [], [], [], [], []]
Node 5 will perform the protocol
[[0], [0, 1], [0, 2], [0, 3], [0, 1, 4], [0, 1, 5], [6], [7], [8], [9]]
[[], [1], [2], [], [1, 4], [1, 5], [], [], [], []]
Node 6 will perform the protocol
[[0], [0, 1], [0, 2], [0, 3], [0, 1, 4], [0, 1, 5], [0, 2, 6], [7], [8], [9]]
[[], [1], [2], [], [1, 4], [1, 5], [2, 6], [], [], []]
Node 7 will perform the protocol
[[0], [0, 1], [0, 2], [0, 3], [0, 1, 4], [0, 1, 5], [0, 2, 6], [0, 3, 7], [8], [9]]
[[], [1], [2], [], [1, 4], [1, 5], [2, 6], [7], [], []]
Node 8 will perform the protocol
[[0], [0, 1], [0, 2], [0, 3], [0, 1, 4], [0, 1, 5], [0, 2, 6], [0, 3, 7], [0, 2, 6, 8], [9]]
[[], [1], [2], [], [1, 4], [1, 5], [2, 6], [7], [2, 6, 8], []]
Node 9 will perform the protocol
[[0], [0, 1], [0, 2], [0, 3], [0, 1, 4], [0, 1, 5], [0, 2, 6], [0, 3, 7], [0, 2, 6, 8], [0, 3, 7, 9]]

Node 0 will perform the protocol
[[0, 1, 4], [0, 1], [0, 2], [0, 3], [0, 1, 4], [0, 1, 5], [0, 2, 6], [0, 3, 7], [0, 2, 6, 8], [0, 3, 7, 9]]
[[0, 1, 4], [1], [2], [], [1, 4], [1, 5], [2, 6], [7], [2, 6, 8], [7, 9]]
Decision is made!
```

Рисунок 4 – Результат работы разработанного алгоритма

На рисунках 5-7 представлена иллюстрация работы алгоритма в виде последовательных состояний сети. Рядом с каждой вершиной обозначены соответственно множества Inc и $NInc$.

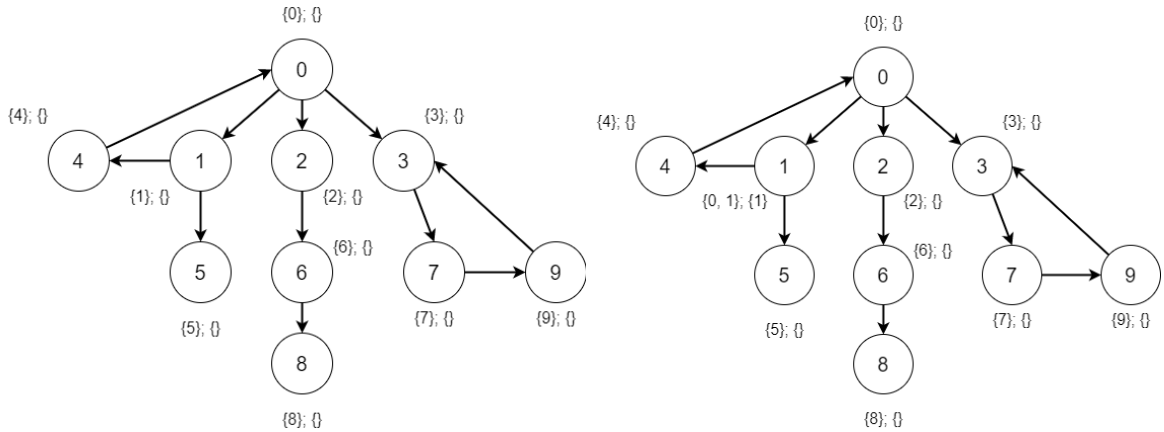

Рисунок 5 – Ход работы алгоритма (шаги 1 и 2)

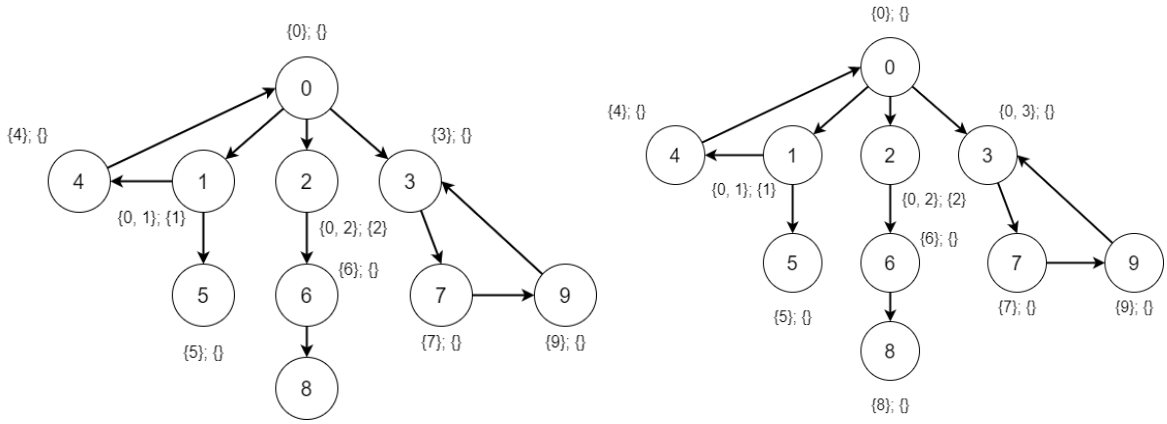

Рисунок 6 – Ход работы алгоритма (шаги 3 и 4)

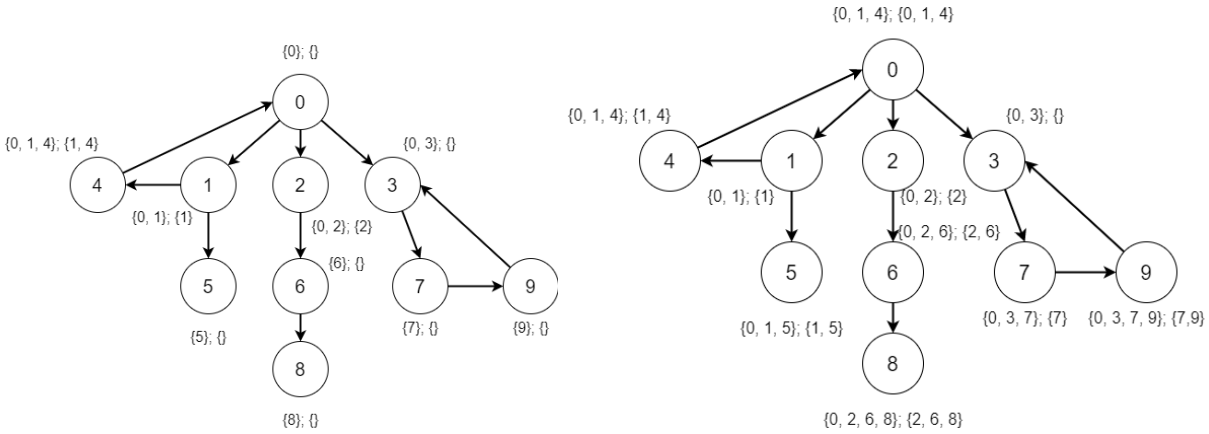

Рисунок 7 – Ход работы алгоритма (шаг 5 и финальное состояние)

Заключение

В ходе выполнения лабораторной работы был изучен волновой алгоритм Финна 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. Рассмотренный алгоритм может использоваться для загрузки и верификации узлов в алгоритмах более высокого уровня.

Код программы

Ниже представлен программный код, отвечающий за выполнение алгоритма:

Листинг 1 – Функция, отвечающая за выполнение алгоритма

```
public void executeEchoFinn(Graph g, Integer start) {
    List<Set<Integer>> Inc = new ArrayList<>();
    List<Set<Integer>> NInc = new ArrayList<>();
    List<List<Map.Entry<Integer, Boolean>>> rec = new ArrayList<>();
    for (var i = 0; i < g.getGraphSize(); i++) {
        Inc.add(Set.of(g.getGraphNodes()[i].getValue()));
        NInc.add(Set.of());

        List<Map.Entry<Integer, Boolean>> r = new ArrayList<>();
        for (var e : g.getPredecessors(i)) {
            r.add(new AbstractMap.SimpleEntry<>(e, Boolean.FALSE));
        }
        rec.add(r);
    }

    int graphSize = g.getGraphSize();
    GraphNode[] nodes = g.getGraphNodes();

    int initiator = start;
    System.out.println("Initiator for graph: " + initiator);

    boolean isEnd = false;
    while (!isEnd) {
        for (int j = 0; j < graphSize; j++) {
            FinnGraphNode finnNode = (FinnGraphNode) nodes[j];

            System.out.println("Node " + j + " will perform the protocol");

            // Ветка для инициатора, должна отработать один раз
            if (j == initiator && !finnNode.isVisited()) {
                ((FinnGraph) g).sentMessagesToNeighbors(j);
            }

            for (var node : finnNode.getNodesFrom()) {
                Set<Integer> IncNew = Stream.concat(Inc.get(j).stream(),
Inc.get(node).stream())
                        .collect(Collectors.toSet());
                Set<Integer> NIncNew = Stream.concat(NInc.get(j).stream(),
NInc.get(node).stream())
                        .collect(Collectors.toSet());

                rec.get(j).stream()
                        .filter(el -> node.equals(el.getKey()))
                        .findFirst()
                        .map(el -> el.setValue(Boolean.TRUE)); // received

                if (rec.get(j).stream()
                        .allMatch(el -> Boolean.TRUE.equals(el.getValue())))
{
                    NIncNew.add(j);
                }

                if (!IncNew.equals(Inc.get(j)) ||
```

```
!NIncNew.equals(NInc.get(j))) {
                    ((FinnGraph) g).sentMessagesToNeighbors(j);
                }

                Inc.set(j, IncNew);
                NInc.set(j, NIncNew);
            }
            finnNode.getNodesFrom().clear();

            if (Inc.get(j).equals(NInc.get(j))) {
                isEnd = true;
                break;
            }

            System.out.println(Arrays.toString(Inc.toArray()));
            System.out.println(Arrays.toString(NInc.toArray()));
        }
    }

    System.out.println(Arrays.toString(Inc.toArray()));
    System.out.println(Arrays.toString(NInc.toArray()));
}
```