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 №3
Тема лабораторной работы	Фазовый волновой алгоритм

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 фазового волнового алгоритма для распределенной сети, представленной в виде графа.

Выполнение лабораторной работы

Фазовый волновой алгоритм является децентрализованным алгоритмом, пригодным для ориентированных сетей с произвольной топологией. Двунаправленные связи должны быть заданы парой параллельных встречных однонаправленных каналов.

В этом алгоритме требуется, чтобы все процессы располагали сведениями о диаметре сети D. Алгоритм будет оставаться корректным (хотя и менее эффективным), если все процессы будут использовать вместо D константу D', превышающую диаметр сети.

Фазовый алгоритм применим для всякой ориентированной сети, по каналам которой осуществляется односторонняя передача сообщений. В этом случае соседями вершины p будут соседи на входе (процессы, которые могут отправлять сообщения процессу p) и соседи на выходе (процессы, которым p может оправлять сообщения). Соседи p на входе образуют множество In_p, а соседи на выходе – множество Out_p.

Основная идея состоит в следующем.

1) Каждый процесс отправляет в точности D сообщений каждому соседу на выходе.
2) Каждому соседу на выходе будет отправлено $(i + 1)$-ое сообщение после того, как от каждого соседа на входе были получены i сообщений.
3) Как только от каждого соседа будет получено в точности D сообщений, процесс завершает алгоритм и принимает решение.

Ниже представлен псевдокод фазового волнового алгоритма:

```
cons D          : integer     = диаметр сети ;
var  Rec_p[q]   : 0..D        init 0 для каждого q ∈ In_p ;
     (* Число сообщений, полученных от q *)
     Sent_p     : 0..D        init 0 ;
     (* Число сообщений, отправленных каждому соседу на выходе
begin if p is initiator then
      begin forall r ∈ Out_p do send ⟨tok⟩ to r ;
            Sent_p := Sent_p + 1 end;
      while min_q Rec_p[q] < D do
            begin receive ⟨tok⟩ (from neighbor q_0) ;
                  Rec_p[q_0] := Rec_p[q_0] + 1 ;
                  if min_q Rec_p[q] ≥ Sent_p and Sent_p < D then
                    begin forall r ∈ Out_p do send ⟨tok⟩ to r ;
                          Sent_p := Sent_p + 1 end
            end;
      decide
end
```

Рисунок 1 - Псевдокод фазового волнового алгоритма

Кроме того, ниже (рис. 2-6) представлена иллюстрация нескольких первых и последнего шагов работы алгоритма на простом графе.

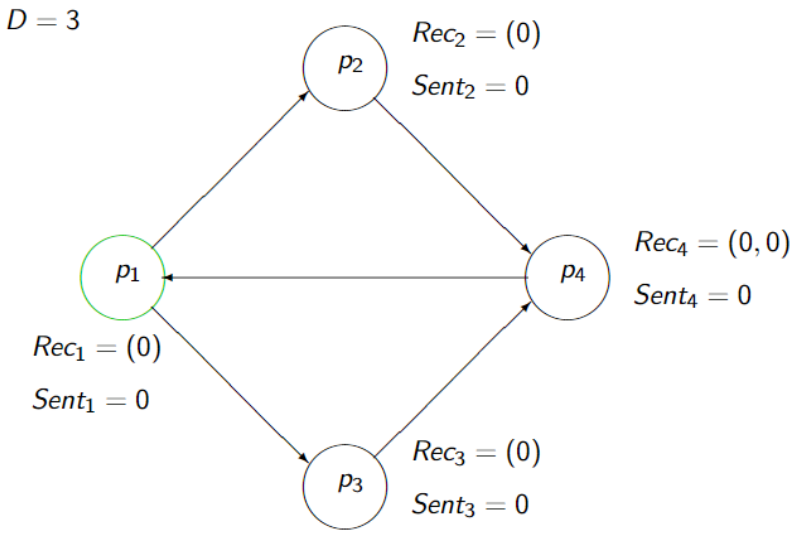

Рисунок 2 – Иллюстрация работы алгоритма, шаг 1

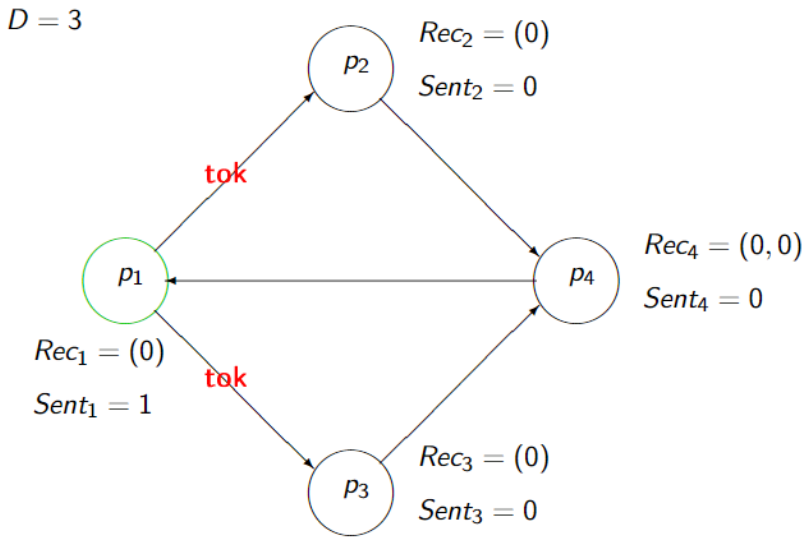

Рисунок 3 – Иллюстрация работы алгоритма, шаг 2

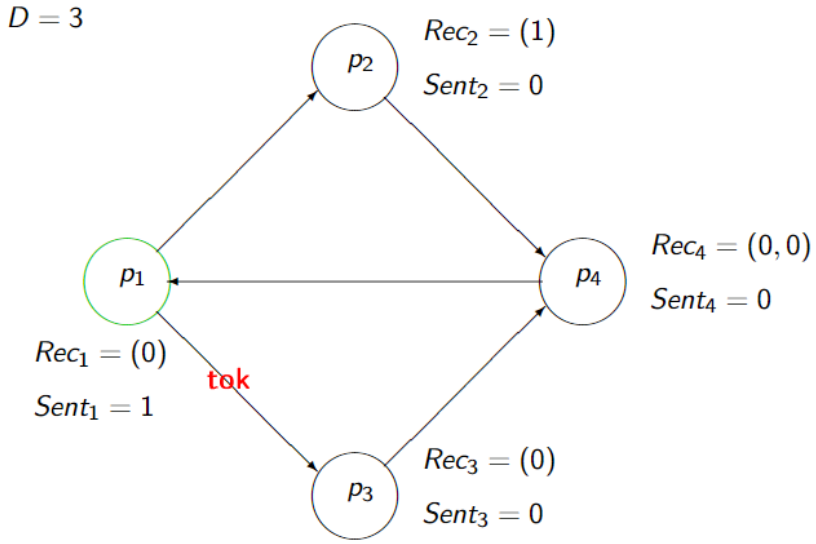

Рисунок 4 – Иллюстрация работы алгоритма, шаг 3

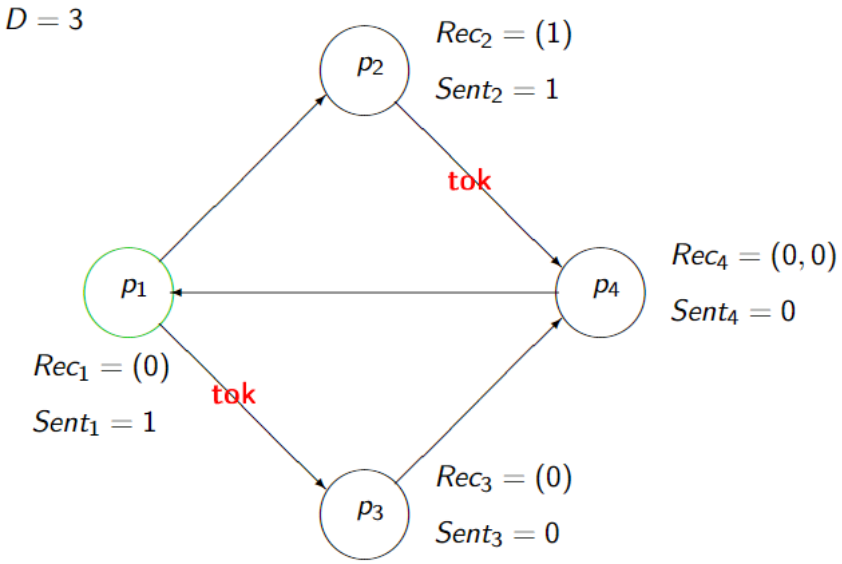

Рисунок 5 – Иллюстрация работы алгоритма, шаг 4

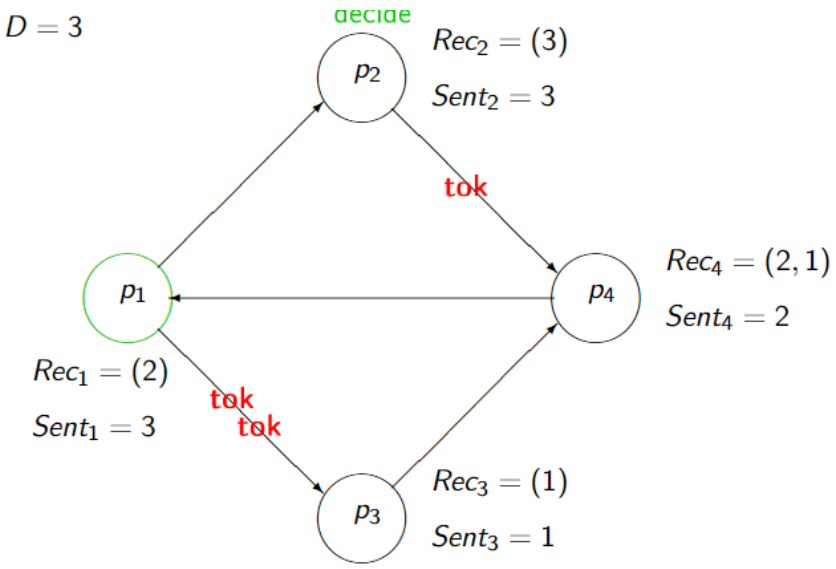

Рисунок 6 – Иллюстрация работы алгоритма, последний шаг

В рамках выполнения лабораторной работы была разработана программа, иллюстрирующая работу фазового волнового алгоритма. Решение производилось на графе, представленном на рисунке 7.

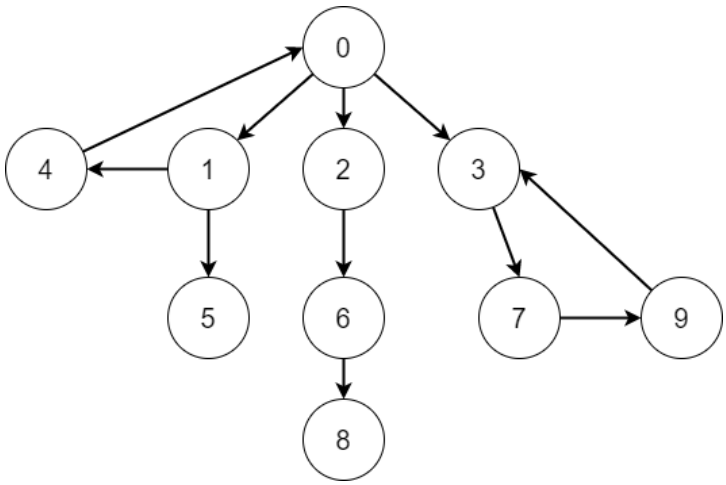

Рисунок 7 – Граф, используемый для решения

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 8 – Полученная программой матрица смежности

Вершиной-инициатором выбирается корневая. Затем происходит рассылка сообщений всем потомкам инициатора. При этом обновляется значение переменной, отвечающей за число отправленных сообщений. Потомки, получая сообщение, в зависимости от минимального количества сообщений, полученных от каждой входящей вершины и числа уже отправленных сообщений посылают сообщения своим потомкам.

Каждая вершина графа в программе представлена потоком, взаимодействие между которыми построено с использованием пакета java.util.concurrency. Потоки общаются через очередь сообщений, которая представлена структурой ConcurrentHashMap, содержащей в качестве ключа идентификатор адресата, а в качестве значения – структуру ConcurrentLinkedQueue, служащую очередью сообщений, с которой работает каждый экземпляр потока. Любой поток, желающий отправить сообщение, кладет его по ключу адресата в соответствующую очередь. Адресат в рабочем цикле считывает сообщения, адресованные ему. При выполнении условия, обозначенного в алгоритме и псевдокоде (рис. 1) происходит отправка сообщений всем потомкам данного процесса. После достижения условия остановки одним из процессов, значение специального volatile флага обновляется так, чтобы рабочие циклы всех потоков тоже завершали свою работу, что, в свою очередь, останавливает работу всего алгоритма.

После того, как поток получает количество сообщений, равное диаметру сети, он принимает решение, работа алгоритма завершается.

На рисунке 9 представлена иллюстрация работы алгоритма для графа, используемого для решения.

```
Initiator for graph: 0
#0; send from 0 to 1; sentCount = 0
#0; send from 0 to 2; sentCount = 0
#0; send from 0 to 3; sentCount = 0
#3; accept from 0; recCount 1; minRecCount 0; sentCount = 0
#3; send from 3 to 7; sentCount = 0
#2; accept from 0; recCount 1; minRecCount 1; sentCount = 0
#2; send from 2 to 6; sentCount = 0
#1; accept from 0; recCount 1; minRecCount 1; sentCount = 0
#1; send from 1 to 4; sentCount = 0
#1; send from 1 to 5; sentCount = 0
#7; accept from 3; recCount 1; minRecCount 1; sentCount = 0
#7; send from 7 to 9; sentCount = 0
#4; accept from 1; recCount 1; minRecCount 1; sentCount = 0
#4; send from 4 to 0; sentCount = 0
#0; accept from 4; recCount 1; minRecCount 1; sentCount = 1
#0; send from 0 to 1; sentCount = 1
#0; send from 0 to 2; sentCount = 1
#0; send from 0 to 3; sentCount = 1
#6; accept from 2; recCount 1; minRecCount 1; sentCount = 0
#6; send from 6 to 8; sentCount = 0
#1; accept from 0; recCount 2; minRecCount 2; sentCount = 1
#1; send from 1 to 4; sentCount = 1
#1; send from 1 to 5; sentCount = 1
#8; accept from 6; recCount 1; minRecCount 1; sentCount = 0
#4; accept from 1; recCount 2; minRecCount 2; sentCount = 1
#4; send from 4 to 0; sentCount = 1
#3; accept from 0; recCount 2; minRecCount 0; sentCount = 1
#2; accept from 0; recCount 2; minRecCount 2; sentCount = 1
#2; send from 2 to 6; sentCount = 1
#5; accept from 1; recCount 1; minRecCount 1; sentCount = 0
#5; accept from 1; recCount 2; minRecCount 2; sentCount = 1
#9; accept from 7; recCount 1; minRecCount 1; sentCount = 0
#9; send from 9 to 3; sentCount = 0
#3; accept from 9; recCount 1; minRecCount 1; sentCount = 1
#3; send from 3 to 7; sentCount = 1
#0; accept from 4; recCount 2; minRecCount 2; sentCount = 2
#0; send from 0 to 1; sentCount = 2
#0; send from 0 to 2; sentCount = 2
#2; accept from 0; recCount 3; minRecCount 3; sentCount = 2
#2; send from 2 to 6; sentCount = 2
#7; accept from 3; recCount 2; minRecCount 2; sentCount = 1
#7; send from 7 to 9; sentCount = 1
#9; accept from 7; recCount 2; minRecCount 2; sentCount = 1
#9; send from 9 to 3; sentCount = 1
#0; send from 0 to 3; sentCount = 2
#3; accept from 0; recCount 3; minRecCount 1; sentCount = 2
#3; accept from 9; recCount 2; minRecCount 2; sentCount = 2
#3; send from 3 to 7; sentCount = 2
#6; accept from 2; recCount 2; minRecCount 2; sentCount = 1
#6; send from 6 to 8; sentCount = 1
#6; accept from 2; recCount 3; minRecCount 3; sentCount = 2
#6; send from 6 to 8; sentCount = 2
#1; accept from 0; recCount 3; minRecCount 3; sentCount = 2
#1; send from 1 to 4; sentCount = 2
#1; send from 1 to 5; sentCount = 2
#7; accept from 3; recCount 3; minRecCount 3; sentCount = 2
#7; send from 7 to 9; sentCount = 2
#4; accept from 1; recCount 3; minRecCount 3; sentCount = 2
#4; send from 4 to 0; sentCount = 2
#8; accept from 6; recCount 2; minRecCount 2; sentCount = 1
#8; accept from 6; recCount 3; minRecCount 3; sentCount = 2
#0; accept from 4; recCount 3; minRecCount 3; sentCount = 3
#0; send from 0 to 1; sentCount = 3
#0; send from 0 to 2; sentCount = 3
#0; send from 0 to 3; sentCount = 3
#3; accept from 0; recCount 4; minRecCount 2; sentCount = 3
#1; accept from 0; recCount 4; minRecCount 4; sentCount = 3
#1; send from 1 to 4; sentCount = 3
#1; send from 1 to 5; sentCount = 3
#9 decide!
#3 decide!
#2 decide!
#5 decide!
#8 decide!
#1 decide!
#7 decide!
#4 decide!
#6 decide!
#0 decide!

Process finished with exit code 0
```

Рисунок 9 – Результат работы разработанного алгоритма

На рисунках 10-13 представлена иллюстрация работы алгоритма в виде последовательных состояний сети. Рядом с каждой вершиной обозначены

соответственно количество отправленных сообщений и число принятых сообщений от каждой входящей вершины для данного процесса.

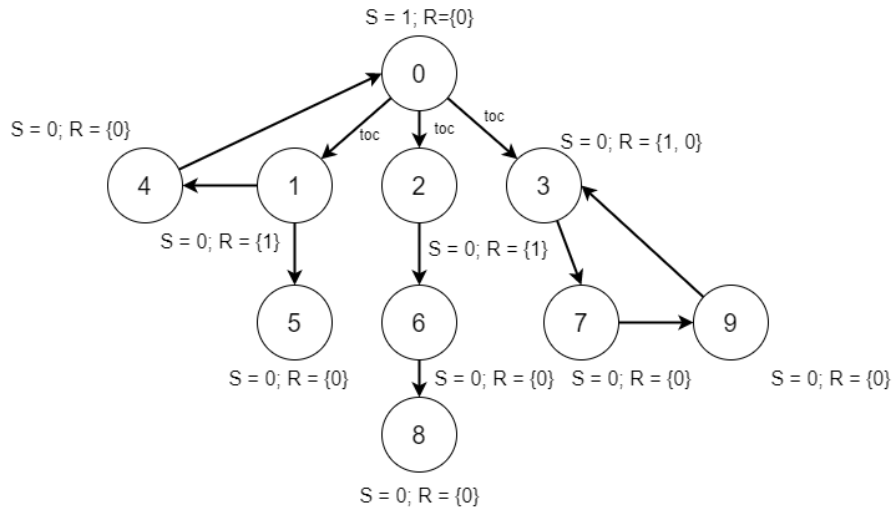

Рисунок 10 – Ход работы алгоритма (шаг 1)

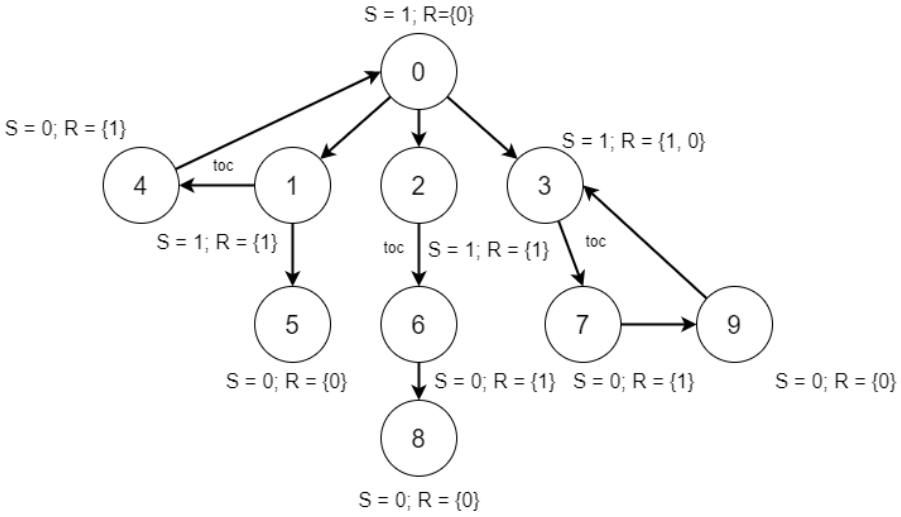

Рисунок 11 – Ход работы алгоритма (шаги 2)

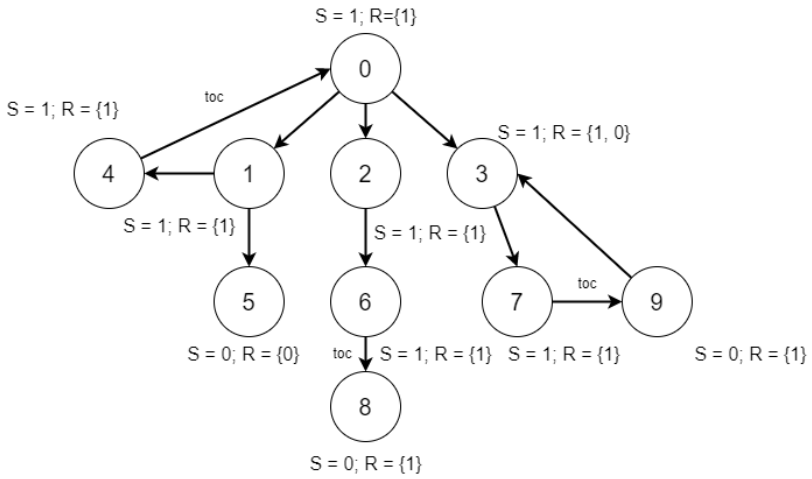

Рисунок 12 – Ход работы алгоритма (шаг 3)

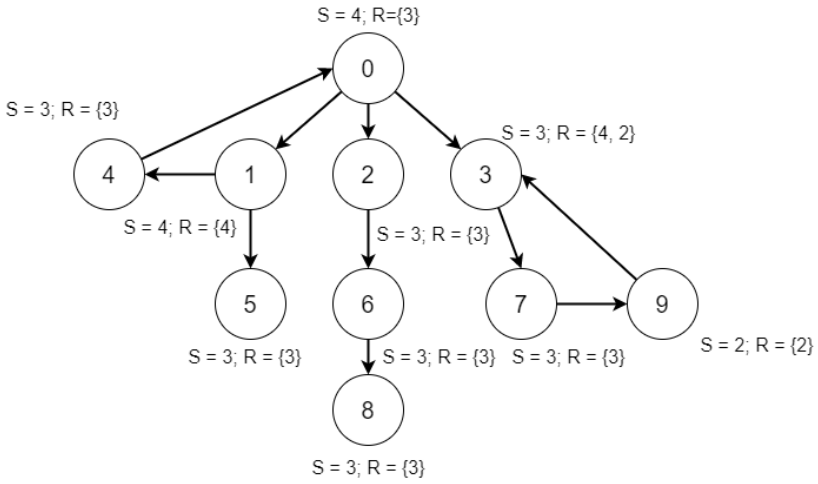

Рисунок 13 – Ход работы алгоритма (шаг 4)

Поскольку программа построена на асинхронном взаимодействии между вершинами, можно видеть по представленному примеру работы алгоритма, что первым решение в данном случае принимает процесс с индексом 1. Хотя в целом такое решение может быть принято процессом, до которого скорее дойдет финальное сообщение, приравнивающее число отправленных и принятых сообщений к значению, служащему условием остановки цикла.

Заключение

В ходе выполнения лабораторной работы был изучен фазовый волновой алгоритм 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.

Код программы

Ниже представлен программный код, отвечающий за выполнение алгоритма, а также некоторые основные функции:

Листинг 1 – Функция, отвечающая за выполнение алгоритма

```
public void executePhaseWave(Graph g, Integer start) {
    System.out.println("Initiator for graph: " + start);
    ((PhaseGraph) g).setInitiator(start);

    ExecutorService taskExecutor =
Executors.newFixedThreadPool(g.getGraphSize());
    Arrays.stream(g.getGraphNodes()).forEach(task ->
taskExecutor.execute((Runnable) task));
    taskExecutor.shutdown();
}

@Override
public void run() {
    if (((PhaseGraph) graph).getInitiator() == this.getValue() &&
!isInitiated) {
        runMain();
        isInitiated = true;
    }
    runWorker();
}

public void runMain() {
    sendMsgToSuccessors();
}

private void sendMsgToSuccessors() {
    successors.forEach(node -> {
        lst.get(node.getValue()).offer(new Message(this, node));

        System.out.println("#" + this.getValue() + "; send from " +
this.getValue() + " to "
                + node.getValue() + "; sentCount = " + sentCount);
    });
    sentCount++;
}

private void runWorker() {
    while (getRecCountMin() < D && !isFinish) {
        Message poll = lst.get(this.getValue()).poll();
        if (poll != null) {
            int value = poll.getFrom().getValue();
            recCount.put(value, recCount.get(value) + 1);
            System.out.println("#" + this.getValue() + "; accept from " +
value
                    + "; recCount " + recCount.get(value)
                    + "; minRecCount " + getRecCountMin()
                    + "; sentCount = " + sentCount);

            if (getRecCountMin() >= sentCount && sentCount < D) {
                sendMsgToSuccessors();
            }
        }
    }
```

```
    isFinish = true;
    System.out.println("#" + this.getValue() + " decide!");
}

private Integer getRecCountMin() {
    return
recCount.values().stream().min(Comparator.comparingInt(Integer::intValue)).or
Else(0);
}
```