Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет

телекоммуникаций и информатики»

**ОТЧËТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1**

по дисциплине

«Теория функционирования распределённых вычислительных систем»

Выполнил:

Студент гр. ИВ-622

Тимофеев Д.А.

Проверила:

Ассистент Кафедры ВС

Ткачева Т.А.

Новосибирск 2020

СОДЕРЖАНИЕ:

[СОДЕРЖАНИЕ: 2](#_Toc38825690)

[1. Цель работы 3](#_Toc38825691)

[2. Теория 5](#_Toc38825692)

[2.1. Производительность 5](#_Toc38825693)

[2.2. Определения основных параметров и основные формулы 5](#_Toc38825694)

[3. Ход работы 7](#_Toc38825695)

[Задание №2.1 7](#_Toc38825696)

[Задание №2.2 7](#_Toc38825697)

[Задание №2.3 8](#_Toc38825698)

[Задание №3.1 8](#_Toc38825699)

[Задание №3.2 9](#_Toc38825700)

[Задание №3.3 9](#_Toc38825701)

[4. Вывод 10](#_Toc38825702)

[5. Листинг 11](#_Toc38825703)

[5.1. Main.java 11](#_Toc38825704)

[5.2. Lab1Grathics.java 11](#_Toc38825705)

1. Цель работы

Имеется распределенная вычислительная система (ВС) укомплектованная *N* одинаковыми элементарными машинами (ЭМ). Основная подсистема (вычислительное ядро) ВС состоит из n ЭМ, *n – N* элементарных машин составляют структурную избыточность. Заданы *λ* – интенсивность потока отказов любой из *N* элементарных машин *([λ] = 1/ч)*, *m* – количество восстанавливающих устройств восстанавливающей системы и *µ* – интенсивность потока восстановления элементарных машин одним восстанавливающим устройством *([µ] = 1/ч)*.

В инженерной практике при анализе надежности ВС наиболее употребительный такие показатели как математическое ожидание времени безотказной работы (средней наработки до отказа) и среднего времени T восстановления ВС, которые равны:

Где R(t) – функция надежности ВС, а U(t) – функция восстановимости ВС.

Для распределенных ВС и T допустимо рассчитывать “частотным” методом [1] который обеспечивает результаты хорошо согласующиеся с более точными вычислениями:

1. Написать программу расчета частотным методом математического ожидания времени Ɵ безотказной работы и среднего времени T восстановления ВС со структурной избыточностью.
2. Построить графики зависимости значений показателя Θ от параметров λ, µ, m и n.
   1. Построить график зависимости Ɵ(n). Параметры: N = 65536; λ = 10-5; m = 1; n = 65527, 65528, …, 65536; µ ∈ {1, 10, 100, 1000}
   2. Построить график зависимости Ɵ(n). Параметры: N = 65536; µ = 1; m = 1; n = 65527, 65528, …, 65536; λ ∈ {10-5, 10-6, 10-7, 10-8, 10-9}.
   3. Построить график зависимости Ɵ(n). Параметры: N = 65536; µ = 1;

λ = 10-5; n = 65527, 65528, …, 65536; m ∈ {1, 2, 3, 4}.

1. Построить графики зависимости значений показателя T от параметров λ, µ, m и n.
   1. Построить график зависимости T(n). Параметры: N = 1000; λ = 10-3; m = 1; n = 900, 910, …, 1000; µ ∈ {1, 2, 4, 6}.
   2. Построить график зависимости T(n). Параметры: N = 8192; µ = 1; m = 1; n = 8092, 8102, …, 8192; λ ∈ {10-5, 10-6, 10-7, 10-8, 10-9}.
   3. Построить график зависимости T(n). Параметры: N = 8192; µ = 1; λ = 10-5; n = 8092, 8102, …, 8192; m ∈ {1, 2, 3, 4}.
2. Теория
   1. Производительность

Современные высокопроизводительные средства обработки информации – распределенные ВС ( distributed computer systems):

* + Большемасштабность ( large-scale), массовый параллелизм(число процессоров
  + Программируемость структуры (structure programmability).
  + Масштабируемость (scalability)
  + Мультипрограммный режим.
  1. Определения основных параметров и основные формулы
  + – интенсивность потока отказов в любой из N машин.
  + - среднее время безотказной работы одной ЭМ
  + m , 1≤m≤N – размер восстанавливающей подсистемы
  + m- количество восстанавливающих устройств
  + µ - интенсивность потока восстановления (1/µ - обнаружение + восстановление).
  + Так же в инженерной практике наиболее употребительны не R(t) и U(t), а математическое ожидание времени безотказной работы ( средняя наработка до отказа и среднее время восстановления:
  + Ꜫ(t) – число исправных машин в момент времени t
  + - вероятность того, что в система начавшей функционировать в состоянии исправных машин
  + R(t) – функция надежности
  + U(t) – функция восстановимости
  + S(t) – Функция готовности
  + Среднее время безотказной работы ВС при n≠N и при n=N следующее:

Среднее время восстановления ВС при *n ≠ 1* и при *n = 1* , следующее

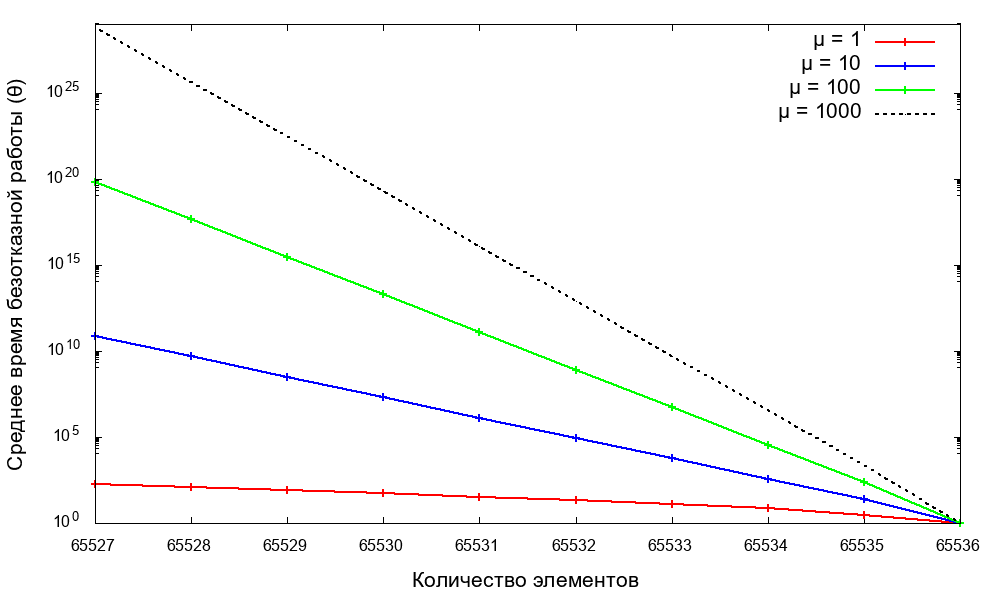
Из этих формул следует следующий вывод:

1. Ход работы

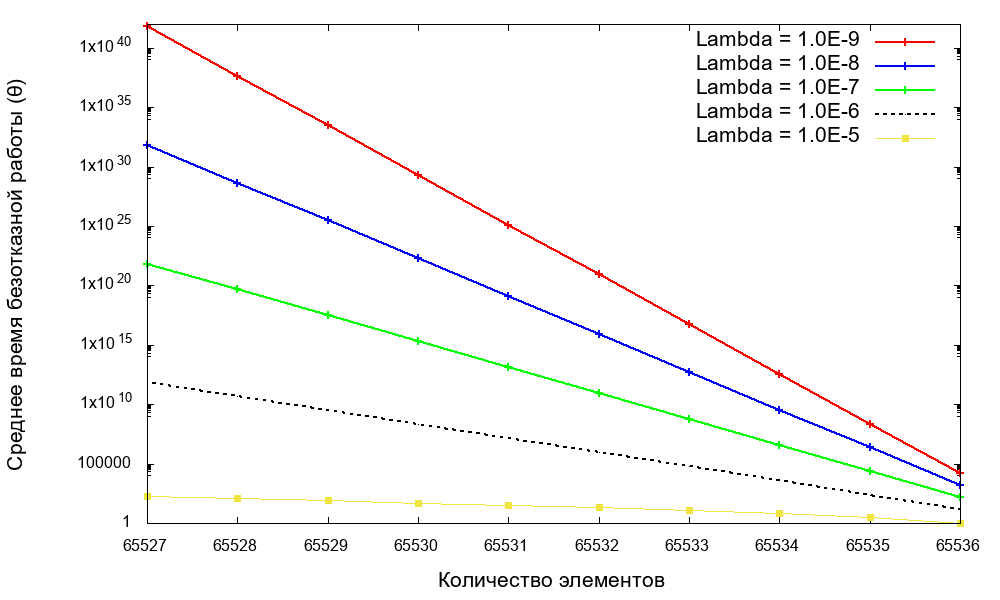
Написал программу.

Провел эксперименты из задания.

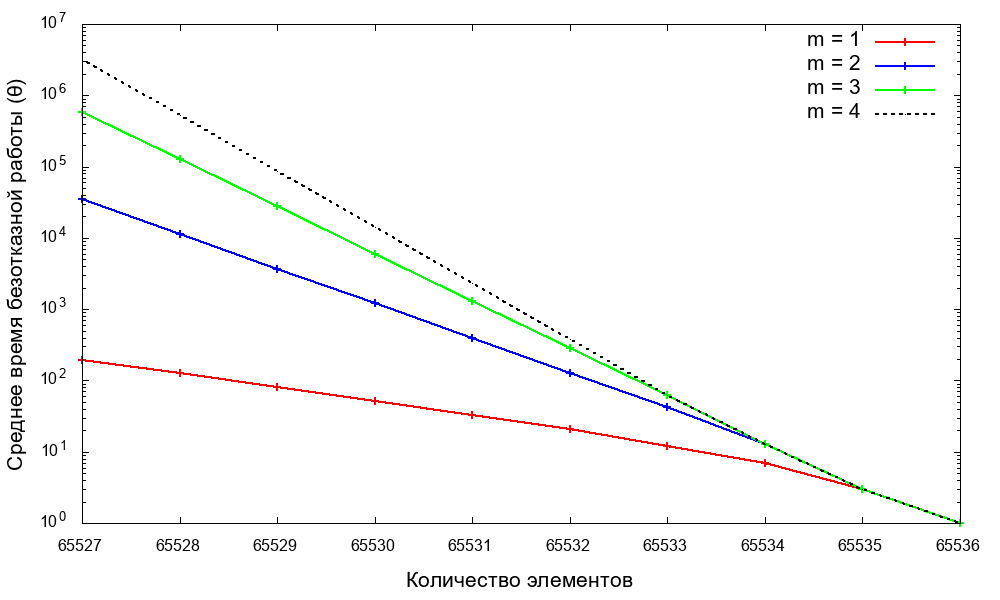
Задание №2.1



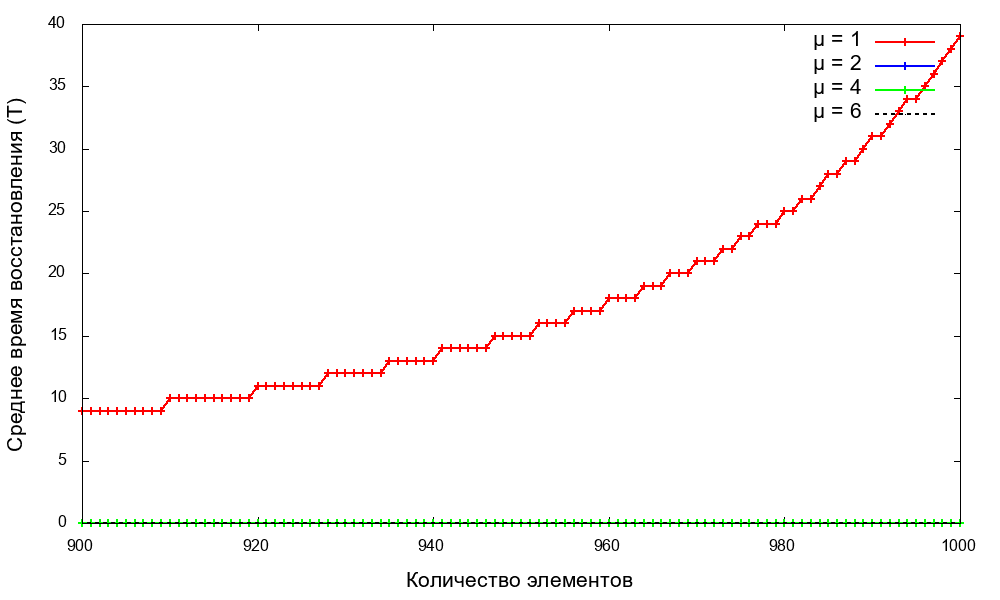
Задание №2.2



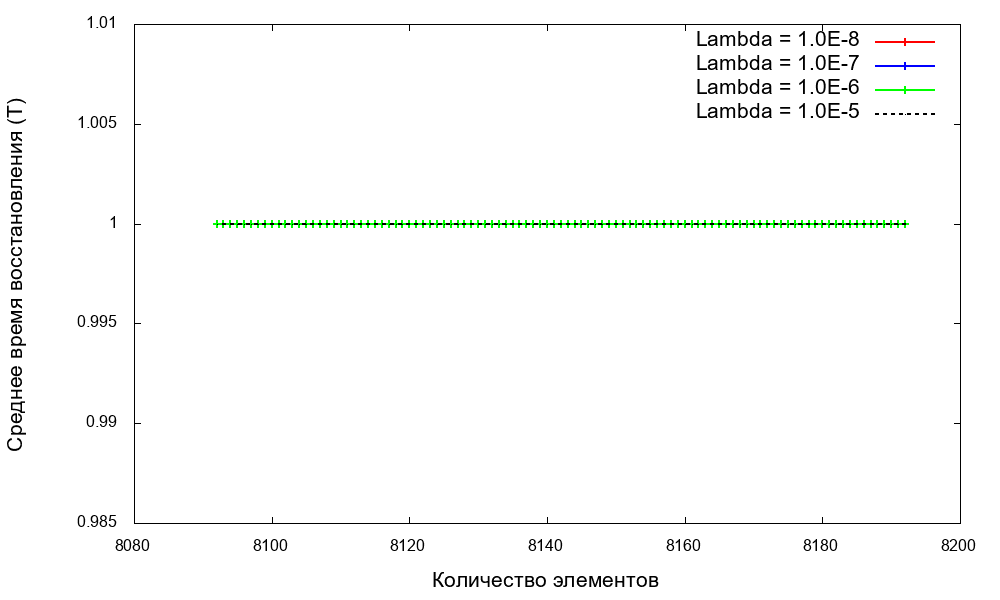
Задание №2.3



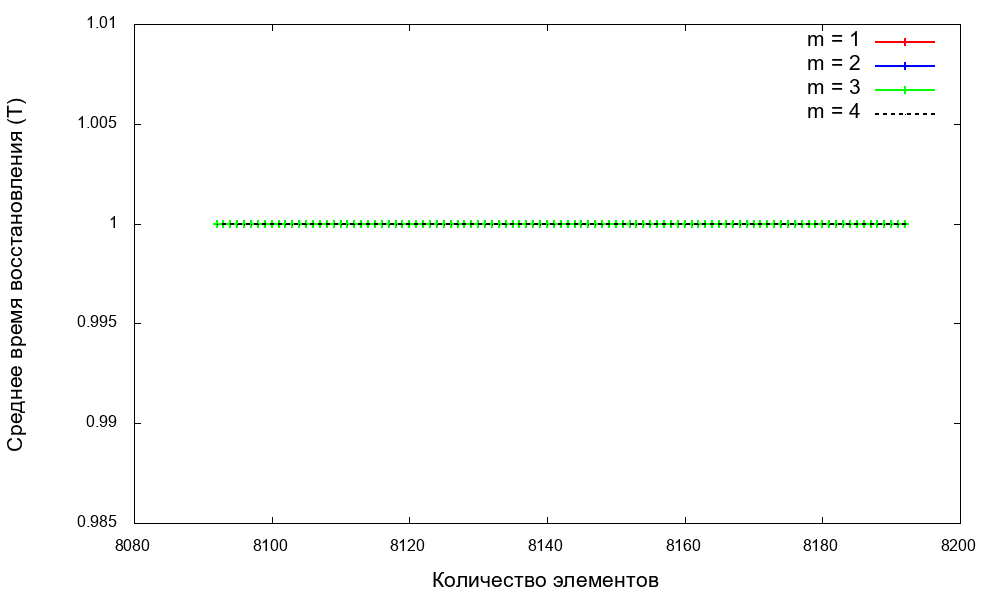
Задание №3.1



Задание №3.2



Задание №3.3



1. Вывод

Большинство параметров, в той или иной форме, связано логарифмической зависимостью (все кроме 3.2 и 3.3).

Следовательно, увеличение восстанавливающей способности системы после определенного количества становится малоэффективной, не рентабельной (это еще раз подтверждается графиками 3.2 и 3.3).

1. Листинг
   1. Main.java

|  |
| --- |
| **package compiler.lab\_1\_usual;  import GFSLibrary.CombinatorCicleClass; import GFSLibrary.ConditionsCicleFor;  import java.io.IOException; import java.io.PrintWriter; import java.util.Map;  public class Main {  public static void main(String[]** args**) throws IOException {  Lab1Grathics lab1GrathicsAverageUptimeTheta = new Lab1Grathics();  lab1GrathicsAverageUptimeTheta.task2p1();  lab1GrathicsAverageUptimeTheta.task2p2();  lab1GrathicsAverageUptimeTheta.task2p3();  lab1GrathicsAverageUptimeTheta.task3p1();  lab1GrathicsAverageUptimeTheta.task3p2();  lab1GrathicsAverageUptimeTheta.task3p3();     }  }** |

* 1. Lab1Grathics.java

|  |
| --- |
| **package compiler.lab\_1\_usual;  import GFSLibrary.StepModificator;  import java.io.IOException; import java.io.PrintWriter;  public class Lab1Grathics {   public enum Formula {  averageTimeRecoveryTau,  averageUptimeTheta  }    private String outputFolderStandtart = "D:\\SibGUTY\_git\\4k2s\\ТФРВС\\labs\\out\_data\\lab\_1\\";    public void task2p1() throws IOException {    int nFrom = 65527;  int nTo = 65537;   int muFromIntensityFloodRecovery = 1;  int muTo = 1001;  StepModificator stepModificator = (**i**)** -> i **\* 10;    int NAllCalculator = 65536;  double lambdalambdaIntendityFloodFailuresMachine = 0.00001;  int m = 1;    String generalTitleFile = "outputAverageUptimeTheta\_2.1";  CreatorFormulaWriterFile creatorFormulaWriterFile = new CreatorFormulaWriterFile(generalTitleFile);  for (int** mu **= muFromIntensityFloodRecovery;** mu **< muTo;** mu **= stepModificator.next(**mu**)) {   String titleLine = String.valueOf(**mu**);  FormulaWriterFile formulaWriterFile =  creatorFormulaWriterFile.createThetaWriterInfrastructure(titleLine, Formula.averageUptimeTheta);   for (int** n **= nFrom;** n **< nTo;** n**++) {  formulaWriterFile.print(lambdalambdaIntendityFloodFailuresMachine,** mu**,** n**, NAllCalculator, m);  }   formulaWriterFile.close();  }   }   public void task2p2() throws IOException {    int nFrom = 65527;  int nTo = 65537;   int lambdaPowFrom = -9;  int lambdaPowTo = -4;    int N = 65536;  int mu = 1;  int m = 1;    String generalTitleFile = "outputAverageUptimeTheta\_2.2";  CreatorFormulaWriterFile creatorFormulaWriterFile = new CreatorFormulaWriterFile(generalTitleFile);  for (int** iPow **= lambdaPowFrom;** iPow **< lambdaPowTo;** iPow**++) {   double lambda = Math.pow(10,** iPow**);   String titleLine = String.valueOf(lambda);  FormulaWriterFile formulaWriterFile =  creatorFormulaWriterFile.createThetaWriterInfrastructure(titleLine, Formula.averageUptimeTheta);   for (int** n **= nFrom;** n **< nTo;** n**++) {  formulaWriterFile.print(lambda, mu,** n**, N, m);  }   formulaWriterFile.close();  }   }   public void task2p3() throws IOException {    int nFrom = 65527;  int nTo = 65537;   int mFrom = 1;  int mTo = 5;    double lambda = Math.pow(10, -5);  int N = 65536;  int mu = 1;    String generalTitleFile = "outputAverageUptimeTheta\_2.3";  CreatorFormulaWriterFile creatorFormulaWriterFile = new CreatorFormulaWriterFile(generalTitleFile);  for (int** m **= mFrom;** m **< mTo;** m**++) {    String titleLine = Integer.toString(**m**);  FormulaWriterFile formulaWriterFile =  creatorFormulaWriterFile.createThetaWriterInfrastructure(titleLine, Formula.averageUptimeTheta);   for (int** n **= nFrom;** n **< nTo;** n**++) {  formulaWriterFile.print(lambda, mu,** n**, N,** m**);  }   formulaWriterFile.close();  }   }   public void task3p1() throws IOException {  int nFrom = 900;  int nTo = 1001;   int muFromIntensityFloodRecovery = 1;  int muTo = 7;  StepModificator stepModificator = (**i**)** -> **{  if (**i **== 1)  return** i **+ 1;  else  return** i **+ 2;  };    int NAllCalculator = 1000;  double lambdalambdaIntendityFloodFailuresMachine = 0.001;  int m = 1;    String generalTitleFile = "outputAverageTimeRecoveryTau\_3.1";  CreatorFormulaWriterFile creatorFormulaWriterFile = new CreatorFormulaWriterFile(generalTitleFile);  for (int** mu **= muFromIntensityFloodRecovery;** mu **< muTo;** mu **= stepModificator.next(**mu**)) {   String titleLine = String.valueOf(**mu**);  FormulaWriterFile formulaWriterFile =  creatorFormulaWriterFile.createThetaWriterInfrastructure(titleLine, Formula.averageTimeRecoveryTau);   for (int** n **= nFrom;** n **< nTo;** n**++) {  formulaWriterFile.print(lambdalambdaIntendityFloodFailuresMachine,** mu**,** n**, NAllCalculator, m);  }   formulaWriterFile.close();  }   }    public void task3p2() throws IOException {  int nFrom = 8092;  int nTo = 8193;   int lambdaPowFrom = -8;  int lambdaPowTo = -4;  StepModificator stepModificator = (**i**)** -> i **+ 1;    int mu = 1;  int NAllCalculator = 8192;  int m = 1;    String generalTitleFile = "outputAverageTimeRecoveryTau\_3.2";  CreatorFormulaWriterFile creatorFormulaWriterFile = new CreatorFormulaWriterFile(generalTitleFile);  for (int** lambdaPow **= lambdaPowFrom;** lambdaPow **< lambdaPowTo;** lambdaPow **= stepModificator.next(**lambdaPow**)) {   double lambdalambdaIntendityFloodFailuresMachine = Math.pow(10,** lambdaPow**);  String titleLine = String.valueOf(lambdalambdaIntendityFloodFailuresMachine);  FormulaWriterFile formulaWriterFile =  creatorFormulaWriterFile.createThetaWriterInfrastructure(titleLine, Formula.averageTimeRecoveryTau);   for (int** n **= nFrom;** n **< nTo;** n**++) {  formulaWriterFile.print(lambdalambdaIntendityFloodFailuresMachine, mu,** n**, NAllCalculator, m);  }   formulaWriterFile.close();  }   }    public void task3p3() throws IOException {  int nFrom = 8092;  int nTo = 8193;   int mFrom = 1;  int mTo = 5;    double lambda = Math.pow(10, -5);  int N = 8192;  int mu = 1;    String generalTitleFile = "outputAverageTimeRecoveryTau\_3.3";  CreatorFormulaWriterFile creatorFormulaWriterFile = new CreatorFormulaWriterFile(generalTitleFile);  for (int** m **= mFrom;** m **< mTo;** m**++) {    String titleLine = Integer.toString(**m**);  FormulaWriterFile formulaWriterFile =  creatorFormulaWriterFile.createThetaWriterInfrastructure(titleLine, Formula.averageTimeRecoveryTau);   for (int** n **= nFrom;** n **< nTo;** n**++) {  formulaWriterFile.print(lambda, mu,** n**, N,** m**);  }   formulaWriterFile.close();  }   }   // <start> <private\_methods>  private class CreatorFormulaWriterFile {  private int numberLine = 1;  private String generalTitleFile;   public *CreatorFormulaWriterFile*(String** generalTitleFile**) {  this.generalTitleFile =** generalTitleFile**;  }    public FormulaWriterFile createThetaWriterInfrastructure(String** titleLine**, Formula** formula**) throws IOException {  PrintWriter output = createFile(generalTitleFile + "-" + numberLine + ".txt");  FormulaWriterFile formulaWriterFile = new FormulaWriterFile(output,** titleLine**,** formula**);   this.numberLine++;   return formulaWriterFile;  }  }    private class FormulaWriterFile {  private PrintWriter output;  Formula formula;    public *FormulaWriterFile*(PrintWriter** output**) {  this.output =** output**;  this.formula = Formula.averageUptimeTheta;  }    public *FormulaWriterFile*(PrintWriter** output**, String** title**, Formula** formula**) {  this.output =** output**;  this.formula =** formula**;   this.output.println(**title**);  }   public void print(double** lambda**, double** mu**, int** n**, int** N**, int** m**) {    double result;  if (this.formula == Formula.averageUptimeTheta)  result = averageUptimeTheta(**lambda**,** mu**,** n**,** N**,** m**);  else if (this.formula == Formula.averageTimeRecoveryTau)  result = averageTimeRecoveryTau(**lambda**,** mu**,** n**,** N**,** m**);  else {  assert (false);  result = -13;  }    output.printf(  "%7d %40.4f ",** n**,  result  );  output.println();  }   public void close() {  this.output.close();  }  }    private double averageUptimeTheta(  double** lambdaIntendityFloodFailuresMachine**,  double** muIntensityFloodRecovery**,  int** nCountCalculator**,  int** NAllCalculator**,  int** mCountReserverCalculator  **) {  double** totalSum **= 0.0f;  double** totalMultiplication **= 1.0f;  for (int** j **=** nCountCalculator **+ 1;** j **<=** NAllCalculator**; ++**j**) {  double multiplication;    boolean jMinus1IncludedInTheRange =** NAllCalculator **-** mCountReserverCalculator **<=** j **- 1 &&** j **- 1 <=** NAllCalculator**;  if (jMinus1IncludedInTheRange) {  multiplication = (**NAllCalculator **- (**j **- 1)) \*** muIntensityFloodRecovery**;  } else {  multiplication =** mCountReserverCalculator **\*** muIntensityFloodRecovery**;  }** totalMultiplication **\*= multiplication / ((**j **- 1) \*** lambdaIntendityFloodFailuresMachine**);** totalSum **+=** totalMultiplication **/ (**j **\*** lambdaIntendityFloodFailuresMachine**);  }  return** totalSum **+ 1 / (**nCountCalculator **\*** lambdaIntendityFloodFailuresMachine**);  }   private double averageTimeRecoveryTau(double** lambda**, double** mu**, int** n**, int** N**, int** m**) {  if (**n **== 1)  return** m **\*** mu**;   double** firstItemMultiplicable **= 1 /** mu**;  for (int** l **= 1;** l **<** n**;** l**++) {** firstItemMultiplicable **\*=** lambda **/** mu**;  }    double** totalSumSecondItem **= 0;  for (int** j **= 1;** j **<** n**;** j**++) {  double** multiplicable **= 1 / (**j **\*** lambda**);   for (int** l **=** j**;** l **<** n**;** l**++)** multiplicable **\*=** l **\*** lambda **/** mu**;** totalSumSecondItem **+=** multiplicable**;  }   return** firstItemMultiplicable **+** totalSumSecondItem**;** */\* if (n == 1)  return m \* mu;   double totalMul = 1.0f;  for (int l = 1; l <= n - 1; ++l)  totalMul \*= l \* lambda / (mu \* l);   double totalSum = 0.0f;  for (int j = 1; j <= n - 1; ++j) {  double totalMul2 = 1.0f;  for (int l = j; l <= n - 1; ++l) {  double mul = (l >= N - m && l <= N) ? (N - l) \* mu : m \* mu;  totalMul2 \*= l \* lambda / mul;  }  totalSum += totalMul2 / (j \* lambda);  }  return totalMul + totalSum;\*/  /\* if (nCountCalculator == 1)  return mCountReserverCalculator \* muIntensityFloodRecovery;   double totalMul = 1.0f;  for (int l = 1; l <= nCountCalculator - 1; ++l)  totalMul \*= l \* lambdaIntendityFloodFailuresMachine / (muIntensityFloodRecovery \* l);   double totalSum = 0.0f;  for (int j = 1; j <= nCountCalculator - 1; ++j) {  double totalMultiplicable = 1.0f;  for (int l = j; l <= nCountCalculator - 1; ++l) {  boolean oneInludedInTheRange = NAllCalculator - mCountReserverCalculator <= l && l <= NAllCalculator;  double multiplicate;  if (oneInludedInTheRange) {  multiplicate = (NAllCalculator - l) \* muIntensityFloodRecovery;  } else {  multiplicate = mCountReserverCalculator \* muIntensityFloodRecovery;  }   totalMultiplicable \*= l \* lambdaIntendityFloodFailuresMachine / multiplicate;  }  totalSum += totalMultiplicable / (j \* lambdaIntendityFloodFailuresMachine);  }  return totalMul + totalSum;\*/* **}   private PrintWriter createFile(String** path**) throws IOException {   PrintWriter f = new PrintWriter(this.outputFolderStandtart +** path**, "UTF-8");   return f;  }  // <end> <private\_methods> }** |