**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. № подл |  |

СОГЛАСОВАНО

Доцент факультета компьютерных наук и департамента больших данных и информационного поиска

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. Л. Чернышев

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

УТВЕРЖДАЮ

Академический руководитель образовательной программы «Программная инженерия», канд. техн. наук, профессор ДПИ ФКН

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

**Программа для построения полинома, соответствующего случайному блужданию на геометрическом графе**

**Пояснительная записка**

**ЛИСТ УТВЕРЖДЕНИЯ**

RU.17701729.504900-01 81 01-1

**Исполнитель**

Студент группы БПИ183

\_\_\_\_\_\_\_\_\_\_\_ / М.И. Сердюков /

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

**Москва 2020**

**УТВЕРЖДЕНО**

RU.17701729.504900-01 81 01-1

**Программа для построения полинома, соответствующего случайному блужданию на геометрическом графе**

**Пояснительная записка**

RU.17701729.504900-01 81 01-1

Листов 20

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. № подл |  |

**Москва 2020**

Содержание

[1. Введение 4](#_Toc40716507)

[1.1. Наименование программы 4](#_Toc40716508)

[1.2. Документы, на основании которых ведется разработка 4](#_Toc40716509)

[2. Назначение и область применения 5](#_Toc40716510)

[2.1. Функциональное назначение 5](#_Toc40716511)

[2.2. Эксплуатационное назначение 5](#_Toc40716512)

[2.3. Краткая характеристика области применения 5](#_Toc40716513)

[3. Технические характеристики 6](#_Toc40716514)

[3.1. Постановка задачи на разработку программы 6](#_Toc40716515)

[3.2. Описание алгоритма и функционирования программы 6](#_Toc40716516)

[3.2.1. Описание алгоритма 6](#_Toc40716517)

[3.2.2. Алгоритм проверки графа на связность 6](#_Toc40716518)

[3.2.3. Алгоритм перебора подграфов 7](#_Toc40716519)

[3.2.4. Алгоритм поиска мостов в графе 7](#_Toc40716520)

[3.3. Описание и обоснование выбора метода организации входных и выходных данных 8](#_Toc40716521)

[3.4. Описание и обоснование выбора состава технических и программных средств 8](#_Toc40716522)

[3.4.1. Состав технических средств 8](#_Toc40716523)

[3.4.2. Состав программных средств 9](#_Toc40716524)

[4. Технико-экономические показатели 10](#_Toc40716525)

[4.1. Предполагаемая потребность 10](#_Toc40716526)

[4.2. Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами. 10](#_Toc40716527)

[5. Список использованных источников 11](#_Toc40716528)

[Приложение 1 12](#_Toc40716529)

[Диаграмма классов 12](#_Toc40716530)

[Приложение 2 13](#_Toc40716531)

[Описание и функциональное назначение классов 13](#_Toc40716532)

[Приложение 3 15](#_Toc40716533)

[Описание и функциональное назначение методов и классов 15](#_Toc40716534)

[ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 20](#_Toc40716535)

1. Введение
   1. Наименование программы

Программа называется – «Программа для построения полинома, соответствующего случайному блужданию на геометрическом графе» (Program for Сonstructing the Polynomial Corresponding to a Random Walk on a Geometric Graph).

Краткое краткое название – «PolymConstr».

* 1. Документы, на основании которых ведется разработка

Приказ декана факультета компьютерных наук И.В. Аржанцева "Об утверждении тем, руководителей курсовых работ студентов образовательной программы «Программная инженерия» факультета компьютерных наук" № 2.3-02/1112-04 от 11.12.2019.

1. Назначение и область применения
   1. Функциональное назначение

Программа решает следующую задачу, по заданному графу строит полином, соответствующий случайному блужданию на метрическом графе.

* 1. Эксплуатационное назначение

Программа предназначена для людей, занимающихся изучением теории графов и комбинаторики.

Эта проблема мотивирована проблемой эволюции волновых пакетов, которые в начальный момент времени локализированы в одной небольшой окрестности.

* 1. Краткая характеристика области применения

Рассмотрим следующую динамическую структуру. Пусть нам дан неориентированный связный граф G(V, E) и некоторая вершина s. В начальный момент времени из вершины s по всем инцидентным ей вершинам начинают двигаться точки. Так же у каждой вершины есть ее время прохождения, сколько время потребуется точке, чтобы пройти по этому ребру. Каждый раз, когда точка достигает вершины u, появляются новые точки, которые теперь движутся по инцидентным ребрам вершины u.

Теперь рассмотрим некоторый момент времени T, число точек движущихся точек в этот момент времени будет равно значению функции N(T) – число точек в момент времени T. Нахождения полинома R(T) приближающего значения функции N(T) и является целью данного программного проекта. Постановка это проблемы была изложена в (V. L. Chernyshev, 2018).

1. Технические характеристики
   1. Постановка задачи на разработку программы

В соответствии с темой курсовой работы – «Программа для построения полинома, соответствующего случайному блужданию на геометрическом графе». Задача состоит в разработке программы для решения задачи – по заданному графу, построить полином, соответствующий случайному блужданию.

* 1. Описание алгоритма и функционирования программы
     1. Описание алгоритма

По материалам (V. L. Chernyshev, 2017) искомый полином R(T) может быть рассчитан как, , где:

|  |  |
| --- | --- |
|  | (1) |
| Где первая сумма по всем связным подграфам G, вторая по всем вершинам v из G, третья по множеству всех различных меток для маршпрутов из s в v. |  |
|  | (2) |

Где первая сумма по всем связным подграфам G, вторая по всем вершинам v из G, которые являются концами мостов, третья по множеству всех различных меток для маршрутов из s в v.

Таким образом, алгоритм вычисления полинома состоит в следующем – перебор всех связных полграфов, перебор вершин в подграфе, рассчет для графа множества всех возможных меток, нахождение мостов в графе. Имея следующий набор алгоритмов расчитатть полином не составит труда.

* + 1. Алгоритм проверки графа на связность

Для проверки графа на связность использовался стандартный алгоритм обхода в глубину (maxdiver).

|  |
| --- |
| private static <E extends Edge<V>, V> void dfs(Graph<E, V> g, V vertex,  DfsVisitor<E, V> visitor, Set<V> used) {  used.add(vertex);  visitor.discoverVertex(vertex);  for (var edge : g.outgoingEdges(vertex)) {  var target = edge.getTarget();  visitor.examineEdge(edge);  if (!used.contains(target)) {  visitor.goEdge(edge);  dfs(g, target, visitor, used);  visitor.returnEdge(edge);  } else {  visitor.backEdge(edge);  }  }  } |
| Листинг 1  На листинге 1 представлен фрагмент реализации функции поиска в глубину, которая используется почти во всех алгоритмах. В ее реализации использовался паттерн программирования Visitor (Visitor), благодаря чему ее очень легко переиспользовать. |

* + 1. Алгоритм перебора подграфов

Для более эффективной реализации перебора графов использовался паттерн программирования Filter (Filter). Класс SubGraphBruteForce отвечает за перебор всех подграфов в заданном графе.

* + 1. Алгоритм поиска мостов в графе

Для перебора вершин в графе, которые являются концами моста использовался алгоритм поиска мостов с использованием поиска в глубину (ИТМО).

|  |
| --- |
| public class BridgesVisitor<E extends Edge<V>, V> implements DfsVisitor<E, V> {  private int time = 0;  private final Set<E> bridges = new HashSet<>();  private final Map<V, V> parent = new HashMap<>();  private final Map<V, Integer> timeIn = new HashMap<>();  private final Map<V, Integer> fup = new HashMap<>();  @Override  public void discoverVertex(V vertex) {  timeIn.put(vertex, time);  fup.put(vertex, time);  ++time;  }  @Override  public void examineEdge(E edge) {  if (edge.getTarget().equals(parent.get(edge.getSource()))) {  return;  }  boolean targetUsed = timeIn.containsKey(edge.getTarget());  if (targetUsed) {  fup.put(edge.getSource(), minTime(edge.getSource(), edge.getTarget()));  }  }  @Override  public void returnEdge(E edge) {  fup.put(edge.getSource(), Math.min(fup.get(edge.getSource()), fup.get(edge.getTarget())));  if (fup.get(edge.getTarget()) > timeIn.get(edge.getSource())) {  bridges.add(edge);  }  }  @Override  public void goEdge(E e) {  parent.put(e.getTarget(), e.getSource());  }  public Set<E> getBridges() {  return bridges;  }  private Integer minTime(V f, V s) {  var first = fup.get(f);  var second = timeIn.get(s);  return Math.min(first, second);  }  }} |
| Листинг 2  На листинге 2 представлен фрагмент реализации Visitor класса для поиска мостов. |

* 1. Описание и обоснование выбора метода организации входных и выходных данных

Пользователь при помощи мышки строит граф, это и является входными данными.

В качестве выходных данных программа рассчитывает полином, соответствующий случайному блужданию в графе.

* 1. **Описание и обоснование выбора состава технических и программных средств**
     1. Состав технических средств

Рекомендуемые характеристики компьютера:

1. Процессор 2.2 ГГц;
2. 8 Гб оперативной памяти;
3. 100 МБ свободного места на компьютере;
4. Клавиатура и компьютерная мышь.
   * 1. Состав программных средств

Для корректной работы программы требуется наличие следующего программного обеспеченья:

1. JDK 11.
2. JavaFX.
3. Интерпретатор python3.
4. Технико-экономические показатели
   1. Предполагаемая потребность

Данный продукт предназначен для исследований в областях теории графов и комбинаторики.

* 1. Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами.

На момент разработки аналогов не было выявлено.

1. Список использованных источников
2. **Correction to the Leading Term of Asymptotics in the Problem of Counting the Number of Points Moving on a Metric Tree** [Отчет] / авт. V. L. Chernyshev A. A. Tolchennikov. - Moscow : [б.н.], 2017.
3. **Intercepting filter pattern** [В Интернете] // Wikipedia. - 01 05 2020 г.. - https://en.wikipedia.org/wiki/Intercepting\_filter\_pattern.
4. **Polynomial approximation for the number of all possible endpoints of a random walk on a metric graph** [Отчет] / авт. V. L. Chernyshev A. A. Tolchennikov. - [б.м.] : ELSEVIER, 2018.
5. **Visitor pattern** [В Интернете] // Wikipedia. - 1 05 2020 г.. - https://en.wikipedia.org/wiki/Visitor\_pattern.
6. **Использование обхода в глубину для поиска мостов** [В Интернете] / авт. ИТМО // Вики-конспекты. - 01 05 2020 г.. - https://neerc.ifmo.ru/wiki/.
7. **Поиск в глубину** [В Интернете] / авт. maxdiver // MAXimal. - 1 5 2020 г.. - https://e-maxx.ru/algo/dfs.

Приложение 1

Диаграмма классов

|  |
| --- |
| https://sun9-30.userapi.com/dNBeBOgTmSugYc7fATNqjheGvbNRpCrh6MRpNA/iAYlTuT8CfI.jpg |
| Рисунок 1 |

Рассмотрим диаграмму классов библиотеки по работе с графами.

На рисунке 1 представлена диаграмма классов. Главными классами являются интерфейс Edge, Graph и DfsVisitor.

Приложение 2

Описание и функциональное назначение классов

Таблица 1. Функциональное назначение классов.

|  |  |
| --- | --- |
| Название класса | Функциональное назначение |
| MainController | Главный контроллер приложения |
| ShowResultController | Контроллер для показа результатов вычислений |
| EditMode | Хранит список состояний для GraphEditor |
| GraphEditor | Инкапсулирует логику работы кон структора графов |
| MathHelper | Вспомогательный класс для математических вычислений |
| PythonScript | Класс для запуска python скриптов |
| Repository | Глобальное хранилище |
| SafeWriter | Безопасный вывод в файл |
| Algorithms | В нем содержится функции для всех алгоритмов |
| BridgesVisitor | Visitor для поиска мостов в графе |
| CompressedGraph | Строит по графу его сжатое представление |
| ConnectedVisitor | Visitor для проверки графа на связность |
| EdgesBruteForce | Класс для перебора подграфов по спискам ребер |
| EdgesListVisitor | Visitor для состовления списка всех ребер графа |
| FilteredIterable | Обертка над коллекцией, производит фильтрацию |
| GraphBuilder | Builder для класса MetricGraph |
| GraphEdge | Реализация интерфейса Edge |
| MarksVisitor | Visitor для нахождения меток графа |
| MetricGraph | Реализация интерфейса Graph |
| ReverseEdgeVisitor | Visitor для нахождения обратных ребер дерева дфс |
| SubGraph | Подграф графа MetricGraph, сам является графом |
| SubGraphBruteForce | Класс для перебора всех подграфов MetricGraph |
| AbstractGraph | Содержит реализацию вывода графа в консоль |
| DfsVisitor | Visitor для обхода графа в глубину |
| Edge | Интерфейс ребра графа |
| ForAllVertices | Функциональный интерфейс для обхода всех ребер графа |
| Graph | Интерфейс графа |
| DialogWindow | Класс создающий диалоговый окна |
| Drawable | Интерфейс для объектов, которые могут быть нарисованы на Canvas |
| Edge2D | Двумерное представление графа ребра |
| GraphCanvas | Специальный холст для изображения графов |
| ResizableCanvas | Холст, который автоматически меняет размеры |
| Vertex2D | Двумерное представление вершины графа |
| Main | Стартовый класс приложения, точка запуска программы |

Приложение 3

Описание и функциональное назначение методов и классов

**public class MainController implements Initializable**

public void initialize(URL location, ResourceBundle resources) – инициализация контроллера.

private void showCountResult() – отображает окно с результатами вычислений

**public class ShowResultController implements Initializable**

public void initialize(URL location, ResourceBundle resources) – инициализация контроллера.

**public class GraphEditor**

public void setEditMode(EditMode mode) – изменяет режим редактирования на заданный.

public boolean hasStartVertex() – проверяет указана ли стартовая вершина.

public GraphEditor(GraphCanvas canvas) – конструктор, принимает холст на котором будет происходить отрисовка.

public void reset() – сбрасывает все данные.

public boolean hasGraph() – проверяет бл ли граф нарисован.

private void addVertex(double x, double y) – добавляет к графу вершину.

private void addEdge(double x, double y) – проводит ребро в графе.

private Vertex2D<Integer> getVertex(double x, double y) – получает вершину, которая находится в координате x, y.

public MetricGraph getGraph() – конструирует получившийся граф.

**public class MathHelper**

private MathHelper() – конструктор приватный.

public static double dist(double x1, double y1, double x2, double y2) – расстояние между точками.

public static double length(double[] a, double[] b) – длина отрезка.

**public class PythonScript**

public static String start(String script) – запускает python скрипт.

public static String startWithArgs(String script, String arg) – запускает python скрипт с аргументом.

**public class SafeWriter**

public static void writeVector(List<Double> vector, String filePath) – записывает вектор Double в файл filePath.

**public class Algorithms**

private Algorithms() – приватный конструктор.

public static <E extends Edge<V>, V> Set<E> findReverseEdges(Graph<E, V> g, V originVertex) – находит множество обратный ребер в графе g.

public static <E extends Edge<V>, V> boolean isConnected(Graph<E, V> g, V originVertex) – проверка связности графа g.

public static <E extends Edge<V>, V> Set<E> findBridges(Graph<E, V> g, V originVertex) – находит все мосты графа g.

public static <E extends Edge<V>, V> Set<E> findEdges(Graph<E, V> g) – находит списко ребер графа g.

public static <E extends Edge<V>, V> Set<E> findMarks(Graph<E, V> g, V originVertex, V targetVertex) – находит множество меток графа g.

public static <E extends Edge<V>, V> void depthFirstSearch(Graph<E, V> g, V v, DfsVisitor<E, V> visitor) – обход графа в глубину.

private static <E extends Edge<V>, V> void dfs(Graph<E, V> g, V vertex, DfsVisitor<E, V> visitor, Set<V> used) – реализация обхода в глубину.

**public class BridgesVisitor<E extends Edge<V>, V> implements DfsVisitor<E, V>**

public void discoverVertex(V vertex) - событие первый раз встречает вершину.

public void examineEdge(E edge) – исследование ребра.

public void returnEdge(E edge) – возвращение по ребру.

public void goEdge(E e) – проход по ребру вперед.

public Set<E> getBridges() – возвращает список мостов.

private Integer minTime(V f, V s) – находит минимум двух величин.

**public class CompressedGraph**

public CompressedGraph(MetricGraph g) – конструктор.

public List<Double> getVector() – возвращает сжатое представление графа.

private void buildCompression() – строит сжатое представление графа.

private void proceedGraph(SubGraph g) – обрабатывает подграф графа.

private void proceedFirstTerm(SubGraph g, Integer vertex, List<Set<GraphEdge>> allMarks) – обработка первого слагаемого.

private void proceedFirstAndSecondTerm(SubGraph g, Integer vertex,

List<Set<GraphEdge>> allMarks, GraphEdge edge) – обработка второго слагаемого.

private List<Set<GraphEdge>> findAllMarksSet(SubGraph g, List<GraphEdge> edges,

Integer targetVertex) – находит множество всех меток для графа g.

**public class ConnectedVisitor<E, V> implements DfsVisitor<E, V>**

public ConnectedVisitor(int vertexNumber) – конструктор.

public void discoverVertex(V vertex) – первый раз встречает вершину.

public boolean isConnected() – возврощает true если граф связен.

**public class EdgesBruteForce**

public EdgesBruteForce(SubGraph g, List<GraphEdge> edges) – конструкто.

public SubGraph next() – переход к следующему подграфу.

**public class EdgesListVisitor<E extends Edge<V>, V> implements DfsVisitor<E, V>**

public void goEdge(E e) – переход по ребру вперед.

public void returnEdge(E e) – переход по ребру назад.

public void backEdge(E e) – просмотр на обратное ребро.

public Set<E> getEdges() – возвращает список всех рёбер графа.

**class FilteredIterable<Value> implements Iterable<Value>**

public FilteredIterable(List<Value> list, Predicate<Value> predicate) – конструктор.

public int size() – размер коллекции.

public Iterator<Value> iterator() – возвращает итератор.

**public class GraphBuilder**

public GraphBuilder() – конструктор.

public GraphBuilder(int vertexNumber) – конструктор от числа вершин.

public void addEdge(int source, int target, double time) – добавить ребро.

public void addVertex(int number) – добавить number вершин.

public void addVertex() – добавить одну вершину.

public void removeEdge(int source, int target) – удалить ребро.

public void removeVertex(int vertex) – удалить вершину.

public void setRoot(int vertex) – установить вершину как корень графа.

public MetricGraph build() – возвращает инстанс графа.

public void reset() – сбрасывает данные.

**public class GraphEdge implements Edge<Integer>**

public GraphEdge(int id, int source, int target, double time) – конструктор.

public int hashCode() – хеш код ребра.

public boolean equals(Object obj) – проверка на равенство.

public Integer getTarget() – возвращает конец ребра.

public Integer getSource() – начало ребра.

**public class MarksVisitor<E extends Edge<V>, V> implements DfsVisitor<E, V>**

public MarksVisitor(V target) – конструктор от вершины.

public Set<E> getMarks() – возвращает множество меток.

public void discoverVertex(V v) – встречает вершину первый раз.

public void goEdge(E e) – проход по ребру вперед.

public void returnEdge(E e) – проход по ребру назад.

public void backEdge(E e) – обратное ребро.

private void markEdge(E edge) – пометить ребро.

**public class MetricGraph extends AbstractGraph<GraphEdge, Integer>**

public MetricGraph(List<GraphEdge> edges, ArrayList<ArrayList<GraphEdge>> list, int root) - конструктор.

public List<GraphEdge> outgoingEdges(Integer vertex) – исходящие ребра из вершины vertex.

public Integer getVertexNumber() – число вершин в графе.

public Integer getEdgesNumber() – число ребер в графе.

public Integer getRoot() – корень графа.

public List<GraphEdge> getEdges() – список ребер графа.

**public class ReverseEdgeVisitor<E extends Edge<V>, V> implements DfsVisitor<E, V>**

public void goEdge(E e) – проход по ребру вперед.

public void returnEdge(E e) – проход по ребру назад.

public void backEdge(E e) – обратное ребро.

public Set<E> getReverseEdges() – возвращает список обратных ребер.

**public class SubGraph extends AbstractGraph<GraphEdge, Integer>**

public SubGraph(MetricGraph otherGraph) – конструктор.

public SubGraph(MetricGraph otherGraph, Predicate<GraphEdge> predicate) – конструктор с предикатом.

public Integer getRoot() – корень графа.

public FilteredIterable<GraphEdge> outgoingEdges(Integer vertex) – исходящие ребра из вершины.

public Integer getVertexNumber() – число вершин в графе.

public Integer getEdgesNumber() – число ребер в графе.

public SubGraph addFilter(Predicate<GraphEdge> filter) – добавляет фильтр.

public List<GraphEdge> getEdges() – список ребер графа.

**public class SubGraphBruteForce**

public SubGraphBruteForce(MetricGraph g) – конструктор.

public SubGraph next() – следующий подграф.

**public abstract class AbstractGraph<E extends Edge<V>, V> implements Graph<E, V>**

public String toString() – строковое представление графа.

**public class DialogWindow**

static public Optional<Double> edgeWeightDialog() – диалоговое окно построение ребра.

static public void errorDialog(String msg) – диалоговое окно с ошибкой.

**public interface Drawable**

void draw(Canvas canvas) – рисует объект на данном холсте.

boolean contains(double x, double y) – проверяет содержит ли объект точку x, y.

**public class Edge2D<Vertex extends Vertex2D> implements Drawable**

public Edge2D(Vertex source, Vertex target, double time) – конструктор.

public void draw(Canvas canvas) – реализация интерфейса Drawable.

public boolean contains(double x, double y) - реализация интерфейса Drawable.

private List<Double> cutLine(double x1, double y1, double x2, double y2, double value) – преобразования векторов.

**public class GraphCanvas extends ResizableCanvas**

public void setData(List<Drawable> data) – задает список данных для отрисовки.

public void draw() – отрисовывает все данные на холсте.

**public class ResizableCanvas extends Canvas**

public boolean isResizable() – переопределение метода.

public double prefHeight(double width) – переопределение метода.

public double minHeight(double width) – переопределение метода.

public double maxHeight(double width) – переопределение метода.

public double minWidth(double height) – переопределение метода.

public double maxWidth(double height) – переопределение метода.

public void resize(double width, double height) – переопределение метода.

public void draw() – метод для отрисоки Canvas.

**public class Vertex2D<Vertex> implements Drawable**

public Vertex2D(Vertex vertex, double x, double y) – конструктор.

public double getX() – возвращает координату X.

public double getY()– возвращает координату Y.

public Vertex getVertex() – вершина.

public void setIsStart(boolean isStart) – делает вершину стартовой.

public void draw(Canvas canvas) – реализация интерфейса Drawable.

public boolean contains(double x, double y) – реализация интерфейса Drawable

public String toString() – строковое представление вершины.

public double centerX() – центр вершины X.

public double centerY() – центр вершины Y.

**public class Main extends Application**

public void start(Stage primaryStage) throws Exception – запуск Stage.

public static void main(String[] args) – точка входа в программу.

# ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| измененных | замененных | новых | аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |