МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

СОЗДАНИЕ ПРИЛОЖЕНИЯ ДЛЯ ОТРИСОВКИ ГРАФОВ И АЛГОРИТМОВ ДЛЯ РАБОТЫ С НИМИ

КУРСОВАЯ РАБОТА

студента 2 курса 211 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Морозова Андрея Денисовича

Научный руководитель
ст. преподаватель _____ М. И. Сафрончик

Заведующий кафедрой
к. ф.-м. н., доцент _____ А. С. Иванов

Саратов 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью настоящей работы является изучение работы фреймворка для кроссплатформенной разработки "Flutter"и разработка приложения для создания графов и взаимодействия с ними.

Поставлены задачи:

— разбор алгоритмов на графах
— разбор работы с Flutter
— построение приложения

1 Введение

1.1 Графы

Граф — математический объект, состоящий из двух множеств. Одно из них — любое конечное множество, его элементы называются *вершинами* графа. Другое множество состоит из пар вершин, эти пары называются *ребрами* графа. [1]

1.2 Основные определения

Ориентированный граф определяется как пара (V, E), где V — конечное множество, а E — бинарное отношение на V, т. е. подмножество множества $V \times V$. Ориентированный граф для краткости называют **орграфом**. Множетсво V называют **множеством вершин графа**, а его элемент называют **вершиной** графа. Множество E называют **множеством рёбер**, а его элементы называют **рёбрами**. Граф может содержать **рёбра-циклы**, соединяющие вершину с собой. На рисунке 1 изображен оринетированный граф с множеством вершин $\{0, 1, 2, 3, 4\}$. 2

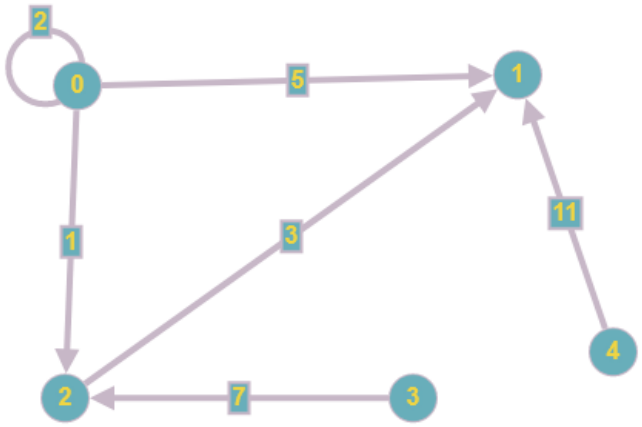

Рисунок 1 – Пример орграфа

В **неориентированном** графе $G = (V, E)$ множество ребер состоит из **неупорядоченных** пар вершин: парами являются множества $\{u, v\}$, где $u, v \in V$ и $u \neq v$. Для неориентированного графа $\{u, v\}$ и $\{v, u\}$ обозначают одно и то же ребро. Неориентированный граф не может содержать рёбер-циклов, и каждое ребро состоит из двух различных вершин. На рисунке 2 изображен неориентированный граф с множеством вершин $\{0, 1, 2, 2, 4\}$

Вершина v **смежна** с вершиной u, если в графе имеется ребро $\{u, v\}$.

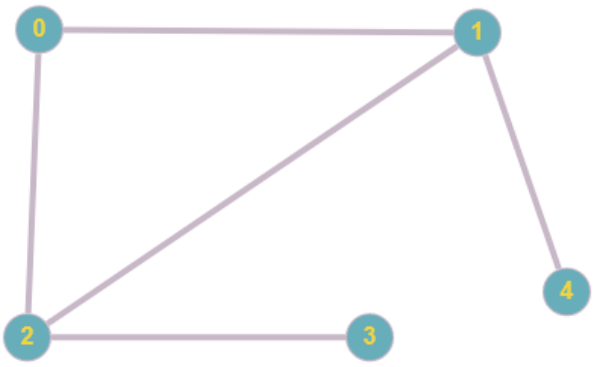

Рисунок 2 – Пример неориентированного графа

Для неориентированных графов отношение смежности является симметричным, но для ориентированных графов это не обязательно.

Степенью вершины в неориентированном графе называется число инцидентных ей рёбер. Для ориентированного графа различают исходящую степень, определяемую как число выходящих из неё рёбер, и входящую степень, определяемую как число входящих в неё рёбер. Сумма исходяшей и входящей степеней называется степенью вершины.

Маршрутом в графе называется конечная чередующаяся последовательность смежных вершин и ребер, соединяющих эти вершины.

Маршрут называется открытым, если его начальная и конечная вершины различны, в противном случае он называется замкнутым.

Маршрут называется **цепью**, если все его ребра различны. Открытая цепь называется **путем**, если все ее вершины различны.

Замкнутая цепь называется **циклом**, если различны все ее вершины, за исключением концевых.

Граф называется **связным**, если для любой пары вершин существует соединяющий их путь. [3]

1.3 Представление графа

Граф можно описать несколькими способами.

1. Перечисление элементов: 1.3.1.
2. Матрица смежности: 1.3.2.
3. Матрица инцидентности: 1.3.3.
4. Список смежности: 1.3.4.

Рассмотрим на примере графа на рисунке 3.

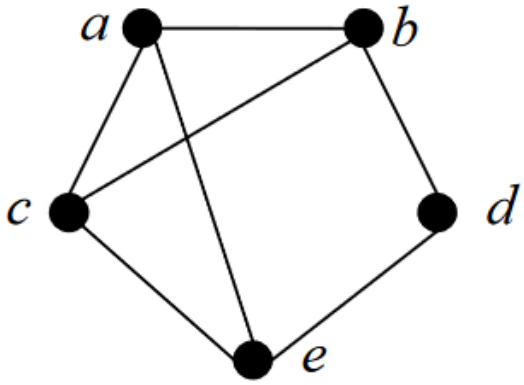

Рисунок 3 – Рассматриваемый граф

1.3.1 Перечисление элементов

Исходя из определения, для того, чтобы задать граф, достаточно перечислить его вершины и ребра (т.е. пары вершин).

Пример:

$$V = \{a, b, c, d, e\}$$

$$E = \{(a, b), (a, c), (a, e), (b, c), (b, d), (c, e), (d, e)\}$$

1.3.2 Матрица смежности

Пусть G — граф с n вершинами, пронумерованными числами от 1 до n. **Матрица смежности** — это таблица с n строками и n столбцами, в которой элемент, стоящий на пересечении строки с номером i и столбца с номером j, равен 1, если вершины с номерами i и j смежны, и 0, если они не смежны.

Таблица 1 – Пример матрицы смежности

0	1	1	0	1
1	0	1	1	0
1	1	0	0	1
0	1	0	0	1
1	0	1	1	0

1.3.3 Матрица инцидентности

Пусть G — граф с вершинами, пронумерованными числами от 1 до n, и ребрами, пронумеровнными от 1 до m. В матрице инцидентности строки соответствуют вершинам, а столбцы рёбрам. На пересечении строки с номером

i и столбца с номером j стоит 1, если вершина с номером i инцидентна ребру с номером j, и 0 в противном случае.

Таблица 2 – Пример матрицы инцидентности

1	1	1	0	0	0	0
1	0	0	1	1	0	0
0	1	0	1	0	1	0
0	0	0	0	1	0	1
0	0	1	0	0	1	1

1.3.4 Списки смежности

Списки смежности часто используются для компьютерного представления графов. Для каждой вершины задается список всех смежных с ней вершин. В структурах данных, применяемых в программировании, списки смежности могут быть реализованы как массив линейных списков.

Указывается номер или имя вершины и перечисляются все смежные с ней вершины.

Пример:

1 : 2, 3, 5

2 : 1, 3, 4

3 : 1, 2, 5

4 : 2, 5

5 : 1, 3, 4

2 Инструменты

Рассмотрим используемый язык и библиотеку для отрисовки.

2.1 Dart

В качестве основы используется язык **Dart**, разработанный компанией Google, и широко используемый для кросс-платформенной разработки [4].

2.2 Flutter

3 Реализация

3.1 Текст с формулами и леммой

3.2 Название другого подраздела

3.2.1 Более мелкий подраздел

3.2.2 Текст с таблицей

Таблица 3 – Результат сокращения словарей неисправностей при помощи масок

1	2	3	4	5	6	7	8
S298	177	1932	341964	61	10797	3,16%	0,61
S344	240	1397	335280	59	14160	4,22%	0,53
S349	243	1474	358182	62	15066	4,21%	0,60
S382	190	12444	2364360	55	10450	0,44%	3,78
S386	274	2002	548548	91	24934	4,55%	1,40
S400	194	13284	2577096	58	11252	0,44%	4,28
S444	191	13440	2567040	60	11460	0,45%	4,26
S510	446	700	312200	70	31220	10,00%	0,63
S526	138	13548	1869624	38	5244	0,28%	2,41
S641	345	5016	1730520	132	45540	2,63%	7,06
S713	343	3979	1364797	131	44933	3,29%	5,61
S820	712	21185	15083720	244	173728	1,15%	126,99
S832	719	21603	15532557	253	181907	1,17%	135,18
S953	326	322	104972	91	29666	28,26%	0,27
S1423	293	750	219750	93	27249	12,40%	0,57
S1488	1359	22230	30210570	384	521856	1,73%	541,69

3.2.3 Текст с кодом программы

ЗАКЛЮЧЕНИЕ

В настоящей работы приведен пример оформления студенческой работы средствами системы LaTeX.

Показано, как можно оформить документ в соответствии:

— с правилами оформления курсовых и выпускных квалификационных работ, принятых в Саратовском государственном университете в 2012 году;
— с правилами оформления титульного листа отчета о прохождении практики в соответствии со стандартом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Захарова, В. Е.* ТЕОРИЯ ГРАФОВ: Учебное пособие / В. Е. Захарова, Д. В. Алексеев. — Нижегородский университет, 2017. — P. 119.

2 *Кормен, Томас Х.* Алгоритмы: построение и анализ, 2-е издание / Томас Х. Кормен, Чарльз И. Лейзерсон, Роналд Л. Ривест, Клиффорд Штайн. — Издательский дом "Вильямс", 2013. — P. 1296.

3 Лекция 45: Алгоритмы на графах. алгоритмы обхода графа. [Электронный ресурс]. — URL: https://intuit.ru/studies/courses/648/504/lecture/11474 (Дата обращения 20.11.2021). Загл. с экр. Яз. рус.

4 Dart overview. [Электронный ресурс]. — URL: https://dart.dev/overview (Дата обращения 20.11.2021). Загл. с экр. Яз. англ.

ПРИЛОЖЕНИЕ А
Код main.dart

Точка входа в программу.

```dart
import 'package:graphs/pages/drawing_page.dart';
import 'package:desktop_window/desktop_window.dart';
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}

Future setupWindow() async {
  await DesktopWindow.setMinWindowSize(const Size(850, 700));
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    setupWindow();
    return const MaterialApp(
      title: "Graphs",
      home: DrawingPage(),
    );
  }
}
```

ПРИЛОЖЕНИЕ Б

Код страницы отрисовки

Описание интерфейса и базовые функции для взаимодействия с информацией.

```dart
import 'package:graphs/src/graph.dart';
import 'package:graphs/src/curve_painter.dart';

import 'package:file_picker/file_picker.dart';
import 'package:flutter/material.dart';

Graphs getGraph() {
  List<Dot> d = <Dot>[];
  d.add(Dot.fromTwoLists("1", [2], [3]));
  d.add(Dot.fromTwoLists("2", [3], [1]));
  d.add(Dot.fromTwoLists("3", [1], [2]));
  return Graphs.fromList("Имя графа", d, true, true);
}

class DrawingPage extends StatefulWidget {
  const DrawingPage({Key? key}) : super(key: key);

  @override
  State<StatefulWidget> createState() => _DrawingPageState();
}

class _DrawingPageState extends State<DrawingPage> {
  double screenSize = 0;
  Graphs graphData = getGraph();
  List<int?>? intListPath;
  List<bool>? dfsAccessTable;
  //List<int?>? dijkstraTable;
  String op = Operations.none;
  int? startDot;
  int? endDot;
  String? dropdownValue1;
  String? dropdownValue2;
  String currOp = "";

  final _textNameController = TextEditingController();
  final _textLnthController = TextEditingController();
  final _textGrNmController = TextEditingController();

  void clearInputData() {
    setState(() {
      _textLnthController.clear();
      _textNameController.clear();
```

```
      dropdownValue1 = null;
      dropdownValue2 = null;
    });
  }

  void clearDropDownVals() {
    setState(() {
      startDot = null;
      intListPath = null;
      dfsAccessTable = null;
      endDot = null;
      currOp = "";
      op = Operations.none;
    });
  }

  // *************buttons**************
  ElevatedButton createButton(String txt, void Function() onPressing) {
    return ElevatedButton(
      onPressed: onPressing,
      style: ButtonStyle(
          backgroundColor: MaterialStateProperty.resolveWith<Color>(
              (states) => Colors.green.shade700)),
      child: Text(txt,
          style: const TextStyle(
            fontSize: 15,
            color: Colors.white70,
            height: 1,
          )),
    );
  }

  void showPopUp(String alertTitle, String err) => showDialog<String>(
        context: context,
        builder: (BuildContext context) => AlertDialog(
          title: Text(alertTitle, style: const TextStyle(fontSize: 26)),
          content: Text(
            err,
            style: const TextStyle(fontSize: 18),
          ),
          actions: <Widget>[
            TextButton(
              onPressed: () => Navigator.pop(context, 'OK'),
              child: const Text('OK'),
            ),
          ],
        ),
      );
```

```
91    // *************buttons*************
92
93    // ***addSpace***
94    SizedBox addSpaceH(double h) {
95      return SizedBox(height: h);
96    }
97
98    SizedBox addSpaceW(double w) {
99      return SizedBox(width: w);
100   }
101   // ***addSpace***
102
103   // *************inputs*************
104   Container createInputBox(String text, double width, IconData? icon,
105       TextEditingController? controller) {
106     if (icon == null) {
107       return Container(
108           width: width,
109           height: 40,
110           margin: const EdgeInsets.symmetric(horizontal: 20, vertical: 5),
111           child: TextField(
112             controller: controller,
113             textAlign: TextAlign.center,
114             onChanged: (name) => graphData.setName(name),
115             decoration: InputDecoration(
116                 contentPadding:
117                     const EdgeInsets.symmetric(vertical: 10, horizontal: 10),
118                 filled: true,
119                 fillColor: Colors.white,
120                 //prefixIcon: Icon(icon, color: Colors.black),
121                 border: const OutlineInputBorder(
122                     borderRadius: BorderRadius.all(Radius.circular(40))),
123                 hintStyle: const TextStyle(color: Colors.black38),
124                 hintText: text),
125           ));
126     }
127     return Container(
128         width: width,
129         height: 40,
130         margin: const EdgeInsets.symmetric(horizontal: 20, vertical: 5),
131         child: TextField(
132           controller: controller,
133           textAlign: TextAlign.center,
134           decoration: InputDecoration(
135               contentPadding:
136                   const EdgeInsets.symmetric(vertical: 10, horizontal: 10),
137               filled: true,
138               fillColor: Colors.white,
```

```
              prefixIcon: Icon(icon, color: Colors.black),
              border: const OutlineInputBorder(
                  borderRadius: BorderRadius.all(Radius.circular(40))),
              hintStyle: const TextStyle(color: Colors.black38),
              hintText: text),
        ));
  }

  SizedBox dropList1(double width) {
    var button = DropdownButton(
      hint: const Text(
        'Select dot',
        style: TextStyle(color: Colors.white, fontSize: 13),
      ),
      alignment: AlignmentDirectional.bottomEnd,
      value: dropdownValue1,
      isDense: true,
      borderRadius: const BorderRadius.all(Radius.circular(20)),
      dropdownColor: Colors.green.shade800,
      style: const TextStyle(
        color: Colors.white,
        fontSize: 18,
      ),
      onChanged: (String? newValue) {
        setState(() {
          dropdownValue1 = newValue;
        });
      },
      items: graphData.getDots().map((location) {
        return DropdownMenuItem(
          child: Text(location.getName()),
          value: location.num.toString(),
        );
      }).toList(),
    );

    return SizedBox(
      child: button,
      width: width,
    );
  }

  SizedBox dropList2(double width) {
    var button = DropdownButton(
      hint: const Text(
        'Select Dot',
        style: TextStyle(color: Colors.white, fontSize: 13),
      ),
```

```
187         alignment: AlignmentDirectional.centerEnd,
188         value: dropdownValue2,
189         isDense: true,
190         borderRadius: const BorderRadius.all(Radius.circular(20)),
191         dropdownColor: Colors.green.shade800,
192         style: const TextStyle(
193           color: Colors.white,
194           fontSize: 18,
195         ),
196         onChanged: (String? newValue) {
197           setState(() {
198             dropdownValue2 = newValue;
199           });
200         },
201         items: graphData.getDots().map((location) {
202           return DropdownMenuItem(
203             child: Text(location.getName()),
204             value: location.num.toString(),
205           );
206         }).toList(),
207       );
208 
209       return SizedBox(
210         child: button,
211         width: width,
212       );
213     }
214   // *************inputs*************
215 
216   //*********ButtonsFunctions*********
217     void addDotPushed() {
218       clearDropDownVals();
219       if (_textNameController.text == "") {
220         showPopUp("Error", "No name in \"Dot name\" box");
221       } else {
222         setState(() {
223           String? res = graphData.addIsolated(_textNameController.text);
224           if (res != null) {
225             showPopUp("Error", res);
226           }
227         });
228       }
229       clearInputData();
230     }
231 
232     void addPathPushed() {
233       clearDropDownVals();
234       if (dropdownValue1 == null) {
```

```
235         showPopUp("Error", "No dot in first box selected");
236       } else if (dropdownValue2 == null) {
237         showPopUp("Error", "No dot in second box selected");
238       } else if (_textLnthController.text == "" && graphData.getUseLengthBool()) {
239         showPopUp("Error", "No length in \"Input length\" box");
240       } else {
241         int? from = int.parse(dropdownValue1!);
242         int? to = int.parse(dropdownValue2!);
243         int? len = int.tryParse(_textLnthController.text);
244         if (len == null && graphData.getUseLengthBool()) {
245           showPopUp("Error",
246               "Can't parse input.\nInts only allowed in \"Input length\"");
247         } else {
248           len ??= 0;
249           setState(() {
250             String? res = graphData.addPath(from, to, len!);
251             if (res != null) showPopUp("Error", res);
252           });
253         }
254       }
255       clearInputData();
256     }
257
258     void changeOriented() {
259       setState(() {
260         String? res = graphData.flipUseOrientation();
261         if (res != null) showPopUp("Error", res);
262       });
263     }
264
265     void changeLength() {
266       setState(() {
267         String? res = graphData.flipUseLength();
268         if (res != null) showPopUp("Error", res);
269       });
270     }
271
272     void delPathPushed() {
273       clearDropDownVals();
274       if (dropdownValue1 == null) {
275         showPopUp("Error", "No dot in first box selected");
276       } else if (dropdownValue2 == null) {
277         showPopUp("Error", "No dot in second box selected");
278       } else {
279         int? from = int.tryParse(dropdownValue1!);
280         int? to = int.tryParse(dropdownValue2!);
281         if (from == null || to == null) {
282           showPopUp("Error",
```

```
283             "Can't parse input.\nInts only allowed in \"Dot number\" and \"Destination number\"");
284       } else {
285         setState(() {
286           String? res = graphData.delPath(from, to);
287           if (res != null) {
288             showPopUp("Error", res);
289           }
290         });
291       }
292     }
293     clearInputData();
294   }
295
296   void delDotPushed() {
297     if (dropdownValue1 != null) {
298       setState(() {
299         String? res = graphData.delDot(int.parse(dropdownValue1!));
300         if (res != null) {
301           showPopUp("Error", res);
302         }
303       });
304     } else {
305       showPopUp("Error", "Nothing in input");
306     }
307     clearDropDownVals();
308     clearInputData();
309   }
310
311   void fileOpener() async {
312     FilePickerResult? result =
313         await FilePicker.platform.pickFiles(allowedExtensions: ["txt"]);
314
315     if (result != null) {
316       if (!result.files.single.path!.endsWith(".txt")) {
317         showPopUp("Error", "Can open only \".txt\" files");
318       } else {
319         setState(() {
320           String? res =
321               graphData.replaceDataFromFile(result.files.single.path!);
322
323           if (res != null) showPopUp("Error", res);
324         });
325       }
326     } else {
327       showPopUp("Error", "No file selected");
328     }
329     clearDropDownVals();
330     clearInputData();
```

```
331     }
332
333     void fileSaver() async {
334       String? outputFile = await FilePicker.platform.saveFile(
335           dialogTitle: 'Please select an output file:',
336           fileName: 'output-file.txt',
337           allowedExtensions: ["txt"]);
338       if (outputFile == null) {
339         showPopUp("Error", "Save cancelled");
340       } else {
341         if (!outputFile.endsWith(".txt")) {
342           outputFile += ".txt";
343         }
344         graphData.printToFile(outputFile);
345       }
346       clearDropDownVals();
347       clearInputData();
348     }
349
350     void bfsPushed() {
351       clearDropDownVals();
352       if (dropdownValue1 == null) {
353         showPopUp("Error", "No dot in first box selected");
354       } else if (dropdownValue2 == null) {
355         showPopUp("Error", "No dot in second box selected");
356       } else {
357         setState(() {
358           startDot = int.parse(dropdownValue1!);
359           endDot = int.parse(dropdownValue2!);
360           currOp = "OP: BFS from $startDot to $endDot";
361           op = Operations.bfs;
362           intListPath = graphData.bfsPath(startDot!, endDot!);
363         });
364         if (intListPath == null) {
365           showPopUp("Info", "There is no path");
366         }
367       }
368       clearInputData();
369     }
370
371     void dfsPushed() {
372       clearDropDownVals();
373       if (dropdownValue1 == null) {
374         showPopUp("Error", "No dot in first box selected");
375       } else {
376         setState(() {
377           startDot = int.parse(dropdownValue1!);
378           op = Operations.dfs;
```

```
379           currOp = "OP: DFS from $startDot";
380           dfsAccessTable = graphData.dfsIterative(startDot!);
381         });
382         if (dfsAccessTable == null) {
383           showPopUp("Err", "report this error.");
384         }
385       }
386
387       clearInputData();
388     }
389
390     void dijkstraPushed() {
391       clearDropDownVals();
392       if (dropdownValue1 == null) {
393         showPopUp("Error", "No number in \"Dot number\" box");
394       } else {
395         setState(() {
396           startDot = int.parse(dropdownValue1!);
397           currOp = "OP: Dijkstra from $startDot";
398           op = Operations.dijkstra;
399           intListPath = graphData.dijkstra(startDot!);
400         });
401         if (intListPath == null) {
402           showPopUp("Err", "report this error.");
403         }
404       }
405       clearInputData();
406     }
407     //*********ButtonsFunctions*********
408
409     // build
410     @override
411     Widget build(BuildContext context) {
412       screenSize = MediaQuery.of(context).size.width;
413       _textGrNmController.text = graphData.getName();
414       return MaterialApp(
415           home: Scaffold(
416         appBar: AppBar(
417             title: const Align(
418                 alignment: Alignment.topLeft,
419                 child: Text("Graph name:\n",
420                     style: TextStyle(
421                       fontSize: 18,
422                       color: Colors.white,
423                     ))),
424             toolbarHeight: 110,
425             flexibleSpace: Container(
426               color: Colors.green.shade900,
```

```
427             child: Column(children: <Widget>[
428               const SizedBox(height: 5),
429               Row(children: [
430                 addSpaceW(screenSize / 8 + 19),
431                 createButton("\nAdd dot\n", addDotPushed),
432                 createInputBox("Dot name", screenSize / 4 - 25, Icons.label,
433                     _textNameController),
434                 addSpaceW(8),
435                 createButton("\nAdd path\n", addPathPushed),
436                 createInputBox("Input length", screenSize / 4 - 25,
437                     Icons.arrow_right_alt_outlined, _textLnthController),
438               ]),
439               addSpaceH(3),
440               Row(children: [
441                 addSpaceW(6),
442                 createInputBox(
443                     "Name", screenSize / 8 - 25, null, _textGrNmController),
444                 createButton("\nDel dot \n", delDotPushed),
445                 addSpaceW(54),
446                 dropList1(screenSize / 4 - 80),
447                 addSpaceW(53),
448                 createButton("\nDel path\n", delPathPushed),
449                 addSpaceW(54),
450                 dropList2(screenSize / 4 - 80),
451               ]),
452             ]),
453           ),
454           actions: [
455             IconButton(
456               onPressed: () {
457                 setState(() {
458                   clearDropDownVals();
459                   graphData.flushData();
460                   clearInputData();
461                 });
462               },
463               icon: const Icon(Icons.delete_sweep),
464               iconSize: 60,
465             ),
466           ]),
467         body: CustomPaint(
468           painter: CurvePainter(
469               graphData: graphData,
470               intListPath: intListPath,
471               dfsAccessTable: dfsAccessTable,
472               start: startDot,
473               end: endDot,
474               op: op),
```

```
475           child: Align(
476             alignment: Alignment.topRight,
477             child: ButtonBar(
478               mainAxisSize: MainAxisSize.min,
479               children: <Widget>[
480                 Text(
481                   currOp,
482                   style: const TextStyle(fontSize: 14, color: Colors.black),
483                 ),
484                 createButton("Bfs", bfsPushed),
485                 createButton("Dfs", dfsPushed),
486                 createButton("Dijkstra", dijkstraPushed),
487                 createButton("Kruskal", () {
488                   clearDropDownVals();
489                   setState(() {
490                     currOp = "OP: Kruscal algo";
491                     graphData.kruskal();
492                   });
493                   clearInputData();
494                 }),
495                 createButton("Clear OP", () {
496                   clearDropDownVals();
497                   clearInputData();
498                 }),
499                 createButton(graphData.getUseLengthStr(), changeLength),
500                 createButton(graphData.getDoubleSidedStr(), changeOriented),
501                 createButton("Save to file", fileSaver),
502                 createButton("Load from file", fileOpener),
503               ],
504             ),
505           ),
506         ),
507       ));
508     }
509 }
```

ПРИЛОЖЕНИЕ В
Код отрисовки графа

Выводит на экран информацияю из графа.

```dart
import 'package:arrow_path/arrow_path.dart';
import 'package:graphs/src/graph.dart';
import 'package:flutter/material.dart';
import 'dart:math';

class Operations {
  static const String bfs = "bfs";
  static const String dfs = "dfs";
  static const String dijkstra = "djkstr";
  static const String none = "none";
}

class CurvePainter extends CustomPainter {
  CurvePainter({
    Key? key,
    required this.graphData,
    required this.intListPath,
    required this.dfsAccessTable,
    required this.start,
    required this.end,
    required this.op,
  });

  List<int?>? intListPath;
  List<bool>? dfsAccessTable;
  String op;
  int? start;
  int? end;
  Graphs graphData;
  final double _dotRad = 7;
  final double _lineWidth = 1.5;
  final Color _lineColor = Colors.black;
  final double _aboveHeight = 5;
  double _circleRad = 100;
  final TextStyle _textStyle = TextStyle(
    color: Colors.red.shade900,
    decorationColor: Colors.green.shade900,
    decorationThickness: 10,
    decorationStyle: TextDecorationStyle.dashed,
    fontSize: 20,
  );
  Map<int, Offset> _off = <int, Offset>{};
  void _drawLine(Canvas canvas, Offset p1, Offset p2) {
    Paint p = Paint();
```

```
45      p.color = _lineColor;
46      p.strokeWidth = _lineWidth;
47      canvas.drawLine(p1, p2, p);
48    }
49
50    void _drawDot(Canvas canvas, Offset p1, [double plusRad = 0, Color? col]) {
51      col ??= Colors.yellow.shade900;
52      var p = Paint();
53      p.color = col;
54      p.strokeWidth = _lineWidth + 2;
55      canvas.drawCircle(p1, _dotRad + plusRad, p);
56    }
57
58    void _drawSelfConnect(Canvas canvas, Offset p1) {
59      var p = Paint();
60      p.color = _lineColor;
61      p.strokeWidth = _lineWidth;
62      p.style = PaintingStyle.stroke;
63      canvas.drawCircle(Offset(p1.dx + _dotRad + 20, p1.dy), _dotRad + 20, p);
64    }
65
66    TextSpan _getTextSpan(String s) => TextSpan(text: s, style: _textStyle);
67    TextPainter _getTextPainter(String s) => TextPainter(
68        text: _getTextSpan(s),
69        textDirection: TextDirection.ltr,
70        textAlign: TextAlign.center);
71
72    void _drawDotNames(Canvas canvas, Offset place, String s) {
73      var textPainter = _getTextPainter(s);
74      textPainter.layout();
75      textPainter.paint(
76          canvas,
77          Offset((place.dx - textPainter.width),
78              (place.dy - textPainter.height) - _aboveHeight));
79    }
80
81    void _drawDotNum(Canvas canvas, Offset size, String s) {
82      var textPainter = TextPainter(
83          text: TextSpan(
84              text: s,
85              style: const TextStyle(
86                color: Colors.black,
87                fontSize: 17,
88              )),
89          textDirection: TextDirection.ltr,
90          textAlign: TextAlign.center);
91      textPainter.layout();
92      textPainter.paint(
```

```
93          canvas,
94          Offset((size.dx - textPainter.width) + 25,
95              (size.dy - textPainter.height) + _aboveHeight + 30));
96    }
97
98    int _getHighInputConnections() {
99      if (graphData.getDots().length != 1 && graphData.getDots().length <= 3) {
100       return -1;
101     }
102     int higest = -1;
103     for (var i in graphData.getDots()) {
104       if (i.getL().length > higest) higest = i.num;
105     }
106     return higest;
107   }
108
109   Map<int, Offset> _getDotPos(int dotsAm, Size size) {
110     Map<int, Offset> off = <int, Offset>{};
111     var width = size.width / 2;
112     var height = size.height / 2;
113     int add = 0;
114     int h = _getHighInputConnections();
115     for (int i = 0; i < dotsAm; i++) {
116       if ((i + 1) != h) {
117         double x =
118             cos(2 * pi * (i - add) / (dotsAm - add)) * _circleRad + width;
119         double y =
120             sin(2 * pi * (i - add) / (dotsAm - add)) * _circleRad + height;
121
122         off[i + 1] = Offset(x, y);
123       } else if ((i + 1) == h) {
124         off[i + 1] = Offset(width + 2, height - 2);
125         add = 1;
126         h = 0;
127       } else {
128         print("GetDotPos error");
129       }
130     }
131
132     return off;
133   }
134
135   void _drawHArrow(Canvas canvas, Size size, Offset from, Offset to,
136       [bool doubleSided = false]) {
137     Path path;
138
139     // The arrows usually looks better with rounded caps.
140     Paint paint = Paint()
```

```
141         ..color = Colors.black
142         ..style = PaintingStyle.stroke
143         ..strokeCap = StrokeCap.round
144         ..strokeJoin = StrokeJoin.round
145         ..strokeWidth = _lineWidth;
146 
147       var length = sqrt((to.dx - from.dx) * (to.dx - from.dx) +
148           (to.dy - from.dy) * (to.dy - from.dy));
149 
150       /// Draw a single arrow.
151       path = Path();
152       path.moveTo(from.dx, from.dy);
153       path.relativeCubicTo(
154           0,
155           0,
156           -(from.dx + to.dx + length) / (length) - 40,
157           -(from.dy + to.dy + length) / (length) - 40,
158           to.dx - from.dx,
159           to.dy - from.dy);
160       path =
161           ArrowPath.make(path: path, isDoubleSided: doubleSided, tipLength: 16);
162       canvas.drawPath(path, paint);
163     }
164 
165     void _drawHighArrow(Canvas canvas, Size size, Offset from, Offset to,
166         [bool doubleSided = false]) {
167       Path path;
168 
169       Paint paint = Paint()
170         ..color = Colors.black
171         ..style = PaintingStyle.stroke
172         ..strokeCap = StrokeCap.round
173         ..strokeJoin = StrokeJoin.round
174         ..strokeWidth = _lineWidth;
175 
176       var length = sqrt((to.dx - from.dx) * (to.dx - from.dx) +
177           (to.dy - from.dy) * (to.dy - from.dy));
178 
179       /// Draw a single arrow.
180       path = Path();
181       path.moveTo(from.dx, from.dy);
182       path.relativeCubicTo(
183           0,
184           0,
185           (from.dx + to.dx) / (length * 2) + 40,
186           (from.dy + to.dy) / (length * 2) + 40,
187           to.dx - from.dx,
188           to.dy - from.dy);
```

```
189       path = ArrowPath.make(
190           path: path,
191           isDoubleSided: doubleSided,
192           tipLength: 13,
193           isAdjusted: false);
194       canvas.drawPath(path, paint);
195     }
196
197     void _drawConnections(
198         Canvas canvas, Size size, List<Dot> dots, Map<int, Offset> off) {
199       for (var i in dots) {
200         var list = i.getL();
201         var beg = off[i.num];
202         for (var d in list.keys) {
203           if (d == i.num) {
204             _drawSelfConnect(canvas, off[d]!);
205           } else {
206             if (graphData.getDoubleSidedBool()) {
207               if (d > i.num) {
208                 _drawHArrow(canvas, size, beg!, off[d]!, false);
209                 if (graphData.getUseLengthBool()) {
210                   _drawDotNames(
211                       canvas,
212                       Offset((off[d]!.dx + beg.dx) / 2 - 18,
213                           (off[d]!.dy + beg.dy) / 2 - 18),
214                       i.getL()[d].toString());
215                 }
216               } else {
217                 _drawHighArrow(canvas, size, beg!, off[d]!, false);
218                 if (graphData.getUseLengthBool()) {
219                   _drawDotNames(
220                       canvas,
221                       Offset((off[d]!.dx + beg.dx) / 2 + 30,
222                           (off[d]!.dy + beg.dy) / 2 + 30),
223                       i.getL()[d].toString());
224                 }
225               }
226             } else {
227               _drawLine(canvas, beg!, off[d]!);
228               if (graphData.getUseLengthBool()) {
229                 _drawDotNames(
230                     canvas,
231                     Offset((off[d]!.dx + beg.dx) / 2, (off[d]!.dy + beg.dy) / 2),
232                     i.getL()[d].toString());
233               }
234             }
235           }
236         }
```

```
237       }
238     }
239
240     void _drawBFS(Canvas canvas) {
241       if (intListPath != null) {
242         for (int i = 0; i < intListPath!.length; i++) {
243           _drawDot(canvas, _off[intListPath![i]]!, 8, Colors.yellow);
244         }
245         _drawDot(canvas, _off[start]!, 9, Colors.green);
246         _drawDot(canvas, _off[end]!, 7, Colors.red.shade200);
247         for (int i = 0; i < intListPath!.length; i++) {
248           _drawDotNum(canvas, _off[intListPath![i]]!, "bfs: #${i + 1}");
249         }
250       }
251     }
252
253     void _drawDFS(Canvas canvas) {
254       if (dfsAccessTable != null) {
255         for (int i = 0; i < dfsAccessTable!.length; i++) {
256           if (dfsAccessTable![i]) {
257             _drawDot(canvas, _off[i + 1]!, 8, Colors.green.shade500);
258             _drawDotNum(canvas, _off[i + 1]!, "dfs: visible");
259           } else {
260             _drawDot(canvas, _off[i + 1]!, 7, Colors.red.shade500);
261             _drawDotNum(canvas, _off[i + 1]!, "dfs: not visible");
262           }
263         }
264         _drawDot(canvas, _off[start]!, 9, Colors.green.shade900);
265       }
266     }
267
268     void _drawDijkstra(Canvas canvas) {
269       if (intListPath != null) {
270         _drawDot(canvas, _off[start]!, 9, Colors.green);
271         for (int i = 0; i < intListPath!.length; i++) {
272           if (intListPath![i] == null) {
273             _drawDotNum(canvas, _off[i + 1]!, "len: INF");
274           } else {
275             _drawDotNum(canvas, _off[i + 1]!, "len: ${intListPath![i]}");
276           }
277         }
278       }
279     }
280
281     @override
282     void paint(Canvas canvas, Size size) {
283       if (size.width > size.height) {
284         _circleRad = size.height / 3;
```

```
285       } else {
286         _circleRad = size.width / 3;
287       }
288
289       _off = _getDotPos(graphData.getDotAmount(), size); //, higest);
290       for (int i in _off.keys) {
291         _drawDot(canvas, _off[i]!);
292       }
293
294       var g = graphData.getDots();
295       switch (op) {
296         case Operations.bfs:
297           {
298             _drawBFS(canvas);
299             break;
300           }
301         case Operations.dfs:
302           {
303             _drawDFS(canvas);
304             break;
305           }
306         case Operations.dijkstra:
307           {
308             _drawDijkstra(canvas);
309             break;
310           }
311         default:
312           {
313             break;
314           }
315       }
316
317       _drawConnections(canvas, size, g, _off);
318       for (int i in _off.keys) {
319         _drawDotNames(
320             canvas, _off[i]!, "${graphData.getDots()[i - 1].getName()}:[$i]");
321       }
322     }
323
324     @override
325     bool shouldRepaint(CustomPainter oldDelegate) {
326       return true;
327     }
328   }
```

ПРИЛОЖЕНИЕ Г

Код класса для работы с графом

Основной класс для хранения и взаимодействия с информацией.

```dart
import 'dart:io';

class Separators {
  static const String dotToConnections = ": ";
  static const String dotToLength = "|";
  static const String space = " ";
  static const String hasLength = "Взвешенный";
  static const String hasNoLength = "НеВзвешенный";
  static const String isOriented = "Ориентированный";
  static const String isNotOriented = "НеОриентированный";
  static const String nL = "\n";
  static const String end = "END";

  Separators();
}

class Dot {
  //Data
  // ignore: prefer_final_fields
  String _name = "";
  int num = -1;
  Map<int, int> _ln = <int, int>{};

  //****Get****
  String getName() => _name;
  bool hasConnection(int n) => _ln.containsKey(n);
  Map<int, int> getL() => _ln;

  int getLength(int x) {
    if (hasConnection(x)) {
      return _ln[x]!;
    }
    return -1;
  }
  //****Get****

  //Set
  void setName(String n) => _name = n;

  //Add
  void addPath(int inp, int length) => _ln[inp] = length;

  //Del
  void delPath(int n) => _ln.removeWhere((key, value) =>
```

```
45        key == n); // удалить обратный путь если не ориентированный
46
47    //Print
48    void printD() {
49      stdout.write("$_name: №$num => ");
50      for (var i in _ln.keys) {
51        stdout.write("$i|${_ln[i]} ");
52      }
53      stdout.write("\n");
54    }
55
56    //******Constructor******
57    Dot([String name = "Undefined", int n = -1]) {
58      _name = name;
59      num = n;
60      _ln = <int, int>{};
61    }
62    Dot.fromTwoLists(String name, List<int> num0, List<int> length,
63        [int n = -1]) {
64      _name = name;
65      num = n;
66      Map<int, int> nw = <int, int>{};
67      if (num0.length != length.length) {
68        print("Error in lists");
69      } else {
70        for (var i = 0; i < num0.length; i++) {
71          nw[num0[i]] = length[i];
72          _ln = nw;
73        }
74      }
75    }
76    Dot.fromMap(String name, Map<int, int> l, [int n = -1]) {
77      _name = name;
78      num = n;
79      _ln = l;
80    }
81    //******Constructor******
82
83    //Copy
84    Dot.clone(Dot a) {
85      _name = a.getName();
86      num = a.num;
87      _ln = a.getL();
88    }
89  }
90
91  class Graphs {
92    static const int intMax = 0x7fffffffffffff;
```

```
93    //Data
94    String _name = "Undefined"; //Имя
95    int _amount = 0; //Количество вершин
96    List<Dot> _dots = <Dot>[]; //Список смежности вершин
97    Map<int, String> _nameTable = <int, String>{}; //Список вершин по именам
98    bool _useLength = false; //Взвешенность
99    bool _oriented = false; //Ориентированность
100
101   //*********************Add************************
102   String? addDot(Dot a) {
103     if (getNumByName(a.getName()) != null) {
104       return ("Dot name \"${a.getName()}\" already in use. Change name or use addPath");
105     }
106     _amount++;
107     a.num = _amount;
108     _dots.add(a);
109     _syncNameTable();
110     checkDots(false);
111     if (!_oriented) _fullFix();
112     return null;
113   }
114
115   bool addDotFromToLists(String name, List<int> num0, List<int> length,
116       [int n = -1]) {
117     var a = Dot.fromTwoLists(name, num0, length, n);
118     if (getNumByName(a.getName()) != null) {
119       print(
120           "Dot name ${a.getName()} already in use. Change name or use addPath");
121       return false;
122     }
123     _amount++;
124     a.num = _amount;
125     _dots.add(a);
126     _syncNameTable();
127     checkDots(false);
128     if (!_oriented) _fixPathAfterInsert(a);
129     return true;
130   }
131
132   String? addIsolated(String name) {
133     var res = addDot(Dot.fromTwoLists(name, [], []));
134     _syncNameTable();
135     return res;
136   }
137
138   String? addPath(int from, int to, [int len = 0]) {
139     if (from <= 0 || from > _amount || to <= 0 && to > _amount) {
140       return "Index out of range. Have dots 1..$_amount";
```

```
141       }
142       _dots[from - 1].addPath(to, len);
143       if (!_oriented) {
144         _dots[to - 1].addPath(from, len);
145       }
146       return null;
147     }
148     //*********************Add************************
149
150     //*********Delete*********
151     String? delPath(int from, int to) {
152       if (from <= 0 || from > _amount || to <= 0 && to > _amount) {
153         return "Can't find specified path";
154       }
155       if (!_dots[from - 1].hasConnection(to)) {
156         return "Already no connection between $from and $to";
157       }
158       _dots[from - 1].delPath(to);
159       if (!_oriented) {
160         _dots[to - 1].delPath(from);
161       }
162       return null;
163     }
164
165     String? delDot(int inn) {
166       if (inn > _amount || inn < 1) {
167         return "Index out of range. Allowed 1..$_amount";
168       }
169       List<int> toDel = <int>[];
170       for (int i in _dots[inn - 1].getL().keys) {
171         toDel.add(i);
172       }
173       for (int i in toDel) {
174         delPath(i, inn);
175       }
176       _dots.removeAt(inn - 1);
177       _syncNum();
178       _syncNameTable();
179       _fixAfterDel(inn);
180       return null;
181     }
182
183     void flushData() {
184       _dots = <Dot>[];
185       _amount = 0;
186       _nameTable = <int, String>{};
187     }
188     //*********Delete*********
```

```
189
190   //******Helper********
191   bool checkDots([bool verbose = false]) {
192     for (var a in _dots) {
193       for (var i in a.getL().keys) {
194         try {
195           if (!_dots[i - 1].getL().containsKey(a.num)) {
196             if (verbose) print("Can't find ${a.num}");
197             return false;
198           }
199         } catch (e) {
200           if (verbose) {
201             print("Can't find Dot $i for path ${a.num}->$i. Exception $e");
202           }
203           _dots[a.num - 1].getL().remove(i);
204           return false;
205         }
206       }
207     }
208     return true;
209   }
210
211   void _fixAfterDel(int inn) {
212     for (int i = 0; i < _dots.length; i++) {
213       Map<int, int> l = <int, int>{};
214       for (int j in _dots[i].getL().keys) {
215         if (j >= inn) {
216           l[j - 1] = _dots[i].getL()[j]!;
217         } else {
218           l[j] = _dots[i].getL()[j]!;
219         }
220       }
221       _dots[i] = Dot.fromMap(_dots[i].getName(), l, _dots[i].num);
222     }
223   }
224
225   void _fixPathAfterInsert(Dot a) {
226     //Для неориентированного
227     for (var i in a.getL().keys) {
228       if (!_dots[i - 1].getL().containsKey(a.num)) {
229         addPath(i, a.num, a.getL()[i]!);
230       }
231     }
232   }
233
234   void _fullFix() {
235     for (var i in _dots) {
236       _fixPathAfterInsert(i);
```

```
237       }
238     }
239
240     void _syncNameTable() {
241       _nameTable = <int, String>{};
242       for (var i in _dots) {
243         _nameTable[i.num] = i.getName();
244       }
245     }
246
247     void _syncNum() {
248       _amount = 0;
249       for (var i in _dots) {
250         i.num = ++_amount;
251       }
252       _syncNameTable();
253     }
254     //******Helper*******
255
256     //*****Setters*******
257     void setName(String name) => _name = name;
258     String? flipUseOrientation() {
259       if (_amount != 0) {
260         return "Can change use of orientation only in empty graph";
261       }
262       _oriented = !_oriented;
263       return null;
264     }
265
266     String? flipUseLength() {
267       if (_amount != 0) {
268         return "Can change use of length only in empty graph";
269       }
270       _useLength = !_useLength;
271       return null;
272     }
273
274     String? replaceDataFromFile(String path) {
275       File file = File(path);
276       List<String> lines = file.readAsLinesSync();
277       if (lines.length < 3) {
278         return "Not enough lines in file";
279       }
280       String name = lines.removeAt(0);
281       bool oriented;
282       switch (lines.removeAt(0)) {
283         case Separators.isOriented:
284           oriented = true;
```

```
285           break;
286         case Separators.isNotOriented:
287           oriented = false;
288           break;
289         default:
290           return "Error on parsing \"IsOriented\"";
291       }
292       bool useLength;
293       switch (lines.removeAt(0).trim()) {
294         case Separators.hasLength:
295           useLength = true;
296           break;
297         case Separators.hasNoLength:
298           useLength = false;
299           break;
300         default:
301           return "Error on parsing \"HasLength\"";
302       }
303       List<Dot> dots = <Dot>[];
304       for (var l in lines) {
305         l = l.trimRight();
306         if (l != Separators.end) {
307           var spl = l.split(Separators.space);
308           List<int> dot = <int>[];
309           List<int> len = <int>[];
310           String name = spl.removeAt(0);
311           name = name.substring(0, name.length - 1);
312           for (var splitted in spl) {
313             if (splitted != "") {
314               var dt = splitted.split(Separators.dotToLength);
315               if (dt.length == 2) {
316                 int? parsed = int.tryParse(dt[0]);
317                 if (parsed == null) {
318                   return "Error while parsing file\nin parsing int in \"${dt[0]}\"";
319                 }
320                 dot.add(parsed);
321                 if (useLength) {
322                   parsed = int.tryParse(dt[1]);
323                   if (parsed == null) {
324                     return "Error while parsing file\nin parsing int in \"${dt[1]}\"";
325                   }
326                   len.add(parsed);
327                 } else {
328                   len.add(0);
329                 }
330               } else if (dt.length == 1) {
331                 int? parsed = int.tryParse(splitted);
332                 if (parsed == null) {
```

```
333                 return "Error while parsing file\nin parsing int in \"$splitted\"";
334               }
335               dot.add(parsed);
336               len.add(0);
337             }
338           }
339         }
340         dots.add(Dot.fromTwoLists(name, dot, len));
341       }
342     }
343     _name = name;
344     _oriented = oriented;
345     _useLength = useLength;
346     _dots = dots;
347     _syncNum();
348     _syncNameTable();
349     if (!_oriented) _fullFix();
350     return null;
351   }
352   //*****Setters*******
353
354   //*****Getters*******
355   bool getDoubleSidedBool() => _oriented;
356   String getDoubleSidedStr() {
357     if (_oriented) return Separators.isOriented;
358     return Separators.isNotOriented;
359   }
360
361   bool getUseLengthBool() => _useLength;
362   String getUseLengthStr() {
363     if (_useLength) return Separators.hasLength;
364     return Separators.hasNoLength;
365   }
366
367   List<Dot> getDots() => _dots;
368   String getName() => _name;
369   String? getNameByNum(int n) => _nameTable[n];
370   Map<int, String> getNameTable() => _nameTable;
371   int getDotAmount() => _dots.length;
372   int? getNumByName(String n) {
373     for (var i in _nameTable.keys) {
374       if (_nameTable[i] == n) return i;
375     }
376     return null;
377   }
378
379   List<List<int>>? getLenTable() {
380     List<List<int>>? out = <List<int>>[];
```

```
    for (int i = 0; i < _amount; i++) {
      List<int> xx = <int>[];
      for (int j = 1; j <= _amount; j++) {
        xx.add(_dots[i].getLength(j));
      }
      out.add(xx);
    }
    return out;
  }

  List<List<int>>? getPathTable() {
    List<List<int>>? out = <List<int>>[];
    for (int i = 0; i < _amount; i++) {
      List<int> xx = <int>[];
      for (int j = 1; j <= _amount; j++) {
        if (_dots[i].getLength(j) != -1) {
          xx.add(i);
        } else {
          xx.add(-1);
        }
      }
      out.add(xx);
    }
    return out;
  }

  List<int>? getLongestPath([int start = 1]) {
    start--;
    if (start < 0 || start >= _amount) {
      return null;
    }
    int max = -1;
    int inD = -1;
    int out = -1;
    List<int>? res = <int>[];
    for (int i = start; i < _amount; i++) {
      var lens = _dots[i].getL();
      for (var d in lens.keys) {
        if (lens[d]! > max) {
          max = lens[d]!;
          inD = i + 1;
          out = d;
        }
      }
    }
    if (inD == -1) {
      return null;
    }
```

```
429       res.add(inD);
430       res.add(out);
431       res.add(max);
432       return res;
433     }
434
435     List<LenDotPath> getSortedPathList() {
436       int max = -1;
437       int inD = -1;
438       int out = -1;
439       List<LenDotPath> result = <LenDotPath>[];
440       for (int i = 0; i < _amount; i++) {
441         var lens = _dots[i].getL();
442         for (var d in lens.keys) {
443           max = lens[d]!;
444           inD = i + 1;
445           out = d;
446           result.add(LenDotPath(max, inD, out));
447         }
448       }
449       result.sort((a, b) => a.l.compareTo(b.l));
450       return result;
451     }
452
453     //*****Getters*******
454
455     //******Print******
456     void printG() {
457       stdout.write("$_name: ");
458       if (_oriented) {
459         stdout.write("Ориентированный, ");
460       } else {
461         stdout.write("Не ориентированный, ");
462       }
463       if (_useLength) {
464         print("Взвешенный");
465       } else {
466         print("Не взвешенный");
467       }
468       for (var i in _dots) {
469         i.printD();
470       }
471     }
472
473     void printToFile(String name) {
474       var file = File(name);
475       file.writeAsStringSync("$_name\n");
476       if (_oriented) {
```

```
477         file.writeAsStringSync("${Separators.isOriented}\n",
478             mode: FileMode.append);
479       } else {
480         file.writeAsStringSync("${Separators.isNotOriented}\n",
481             mode: FileMode.append);
482       }
483       if (_useLength) {
484         file.writeAsStringSync("${Separators.hasLength}\n",
485             mode: FileMode.append);
486       } else {
487         file.writeAsStringSync("${Separators.hasNoLength}\n",
488             mode: FileMode.append);
489       }
490       for (int i = 0; i < _amount; i++) {
491         file.writeAsStringSync(_dots[i].getName() + Separators.dotToConnections,
492             mode: FileMode.append);
493         var d = _dots[i].getL();
494         for (var j in d.keys) {
495           file.writeAsStringSync(
496               j.toString() + Separators.dotToLength + d[j].toString() + " ",
497               mode: FileMode.append);
498         }
499         file.writeAsStringSync(Separators.nL, mode: FileMode.append);
500       }
501       file.writeAsStringSync(Separators.end, mode: FileMode.append);
502     }
503     //******Print******
504
505     //*******Constructor*********
506     Graphs(
507         [String name = "Undefined",
508         bool hasLen = false,
509         bool isOriented = false]) {
510       _name = name;
511       _dots = <Dot>[];
512       _useLength = hasLen;
513       _oriented = isOriented;
514       _amount = 0;
515       _nameTable = <int, String>{};
516     }
517     Graphs.fromList(String name, List<Dot> dots, bool hasLen, bool oriented) {
518       _name = name;
519       _dots = dots;
520       _useLength = hasLen;
521       _amount = _dots.length;
522       _oriented = oriented;
523       _syncNum();
524       if (!_oriented) _fullFix();
```

```
525   }
526   Graphs.fromFile(String path) {
527     replaceDataFromFile(path);
528   }
529   //*******Constructor********
530
531   //Copy
532   Graphs.clone(Graphs a) {
533     _name = a.getName();
534     _dots = a.getDots();
535     _oriented = a.getDoubleSidedBool();
536     _useLength = a.getUseLengthBool();
537     _amount = _dots.length;
538     _syncNameTable();
539   }
540
541   //************Алгоритмы************
542   List<int>? bfsPath(int startDot, int goalDot) {
543     if (startDot == goalDot) return [startDot];
544     //if (!bfsHasPath(startDot, goalDot)) return null;
545     startDot--;
546     goalDot--;
547     List<List<int>>? graph = getLenTable();
548     List<bool> used = <bool>[];
549     List<int> dst = <int>[];
550     List<int> pr = <int>[];
551
552     for (int i = 0; i < _amount; i++) {
553       dst.add(-1);
554       used.add(false);
555       pr.add(0);
556     }
557
558     List<int> q = <int>[];
559     q.add(startDot);
560     used[startDot] = true;
561     dst[startDot] = 0;
562     pr[startDot] =
563         -1; //Пометка, означающая, что у вершины startDot нет предыдущей.
564
565     while (q.isNotEmpty) {
566       int cur = q.removeAt(0);
567       int x = 0;
568       for (int neighbor in graph![cur]) {
569         if (neighbor != -1) {
570           if (!used[x]) {
571             q.add(x);
572             used[x] = true;
```

```
573             dst[x] = dst[cur] + 1;
574             pr[x] = cur; //сохранение предыдущей вершины
575           }
576         }
577         x++;
578       }
579     }
580
581     //Восстановим кратчайший путь
582     //Для восстановления пути пройдём его в обратном порядке, и развернём.
583     List<int> path = <int>[];
584
585     int cur = goalDot; //текущая вершина пути
586     path.add(cur + 1);
587
588     while (pr[cur] != -1) {
589       //пока существует предыдущая вершина
590       cur = pr[cur]; //переходим в неё
591       path.add(cur + 1); //и дописываем к пути
592     }
593
594     path = path.reversed.toList();
595
596     if (path[0] == (startDot + 1) &&
597         path[1] == (goalDot + 1) &&
598         !_dots[startDot].hasConnection(goalDot + 1)) return null;
599     return path;
600   }
601
602   List<bool>? dfsIterative(int v) {
603     v--;
604     List<bool> label = <bool>[];
605     for (int i = 0; i < _amount; i++) {
606       label.add(false);
607     }
608     List<int> stack = <int>[];
609     stack.add(v);
610     //pos.add(v);
611     while (stack.isNotEmpty) {
612       v = stack.removeLast();
613       if (!label[v]) {
614         label[v] = true;
615         for (int i in _dots[v].getL().keys) {
616           stack.add(i - 1);
617           //pos.add(i);
618         }
619       }
620     }
```

```
621       return label;
622     }
623
624     List<int?> dijkstra(int from) {
625       List<int?> d = List<int?>.filled(_amount, intMax);
626       List<int> p = List<int>.filled(_amount, -1);
627
628       d[from - 1] = 0;
629       List<bool> u = List<bool>.filled(_amount, false);
630       for (int i = 0; i < _amount; ++i) {
631         int v = -1;
632         for (int j = 0; j < _amount; ++j) {
633           // int t;
634           if (!u[j] && (v == -1 || d[j]! < d[v]!)) {
635             v = j;
636           }
637         }
638         if (d[v] == intMax) break;
639         u[v] = true;
640         for (int to in _dots[v].getL().keys) {
641           int len = _dots[v].getL()[to]!;
642           if (!_useLength && len == 0) len = 1;
643           if (d[v]! + len < d[to - 1]!) {
644             d[to - 1] = d[v]! + len;
645             p[to - 1] = v;
646           }
647         }
648       }
649       for (int i = 0; i < d.length; i++) {
650         // подумать как убрать эту часть
651         if (d[i] == intMax) d[i] = null;
652       }
653       return d;
654     }
655
656     Graphs? kruskal() {
657       List<LenDotPath> g = getSortedPathList();
658       //int cost = 0;
659       List<Dot> res = <Dot>[];
660       for (int i = 0; i < _amount; i++) {
661         res.add(Dot(_dots[i].getName(), _dots[i].num));
662       }
663       List<int> treeId = List<int>.filled(_amount, 0);
664       for (int i = 0; i < _amount; ++i) {
665         treeId[i] = i;
666       }
667       for (int i = 0; i < g.length; ++i) {
668         int a = g[i].d - 1, b = g[i].p - 1;
```

```
669         int l = g[i].l;
670         if (treeId[a] != treeId[b]) {
671           //cost += l;
672           res[a].addPath(b + 1, l);
673           int oldId = treeId[b], newId = treeId[a];
674           for (int j = 0; j < _amount; ++j) {
675             if (treeId[j] == oldId) {
676               treeId[j] = newId;
677             }
678           }
679         }
680       }
681       _dots = res;
682       return Graphs.fromList(_name, res, _useLength, _oriented);
683     }
684     //************Алгоритмы************
685   }
686
687   class LenDotPath {
688     late int l;
689     late int d;
690     late int p;
691     LenDotPath(int len, dot, path) {
692       l = len;
693       d = dot;
694       p = path;
695     }
696   }
```