

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВАЯ РАБОТА
ЗАЩИЩЕНА С ОЦЕНКОЙ
РУКОВОДИТЕЛЬ

доц., к.ф.-м.н., доцент
должность, уч. степень, звание

подпись, дата

М. В. Фаттахова
инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ

Предприятие «Маяк»

по дисциплине: ПРИКЛАДНЫЕ МОДЕЛИ ОПТИМИЗАЦИИ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4932

подпись, дата

С. И. Коваленко
инициалы, фамилия

Задача 16. Предприятие «Маяк»

Предприятие имеет 11 линий пяти типов. Производительность линий, выпускающих 7 видов продукции, время (часы), за которое на данной линии производится единица продукции, и стоимость 1 ч. работы линии отражены в таблице 1:

Количество	Тип	П1	П2	П3	П4	П5	П6	П7	Стоимость 1 ч. работы, \$
3	Линия 1	0,5	0,7	0,8	0,4	0,7	0,8	0,5	20
3	Линия 2	0,65	0,91	1,04	-	0,91	1,04	0,63	15
2	Линия 3	0,35	0,49	0,56	0,28	0,49	0,56	0,35	30
2	Линия 4	0,25	0,35	0,4	-	0,35	0,4	0,25	40
1	Линия 5	-	0,28	0,32	0,16	0,28	0,32	0,2	50

Таблица 1

Прибыль и потребности рынка в товаре приведены в таблице 2.

	Б1	Б2	Б3	Б4	Б5	Б6	Б7	
Потребность	3500	1000	1000	2000	800	200	1000	шт.
Прибыль	25	26	28	24	27	29	23	\$

Таблица 2

Предприятие может работать в нормальном режиме 16 ч в день и в среднем 24 дня в месяц. Известно также, что постоянные издержки для работающей линии (издержки запуска) составляют 1 тыс. долл. в месяц. Составьте оптимальный план производства и рассчитайте, какую прибыль предприятие сможет получить за 1 месяц.

Математическая модель

x_{ij} – время необходимое для производства i -ого продукта на j -ой линии

y_{ij} – количество i -ого товара произведённого на j -ой линии

b_i – прибыль от продажи i -ого продукта

c_i – потребность рынка в i -ом продукте

e_j – стоимость работы j -ой линии

Целевая функция

$$L = \sum_{i=1}^7 \sum_{j=1}^{11} (y_{ij} \cdot b_i) - \sum_{i=1}^7 \sum_{j=1}^{11} (y_{ij} \cdot x_{ij} \cdot e_j) \rightarrow \max$$

Ограничения

$$y_{ij} \geq 0 \quad i = \overline{1,7}, j = \overline{1,11}$$

$$\sum_{i=1}^7 y_{ij} \cdot x_{ij} \leq 24 \cdot 16 \quad j = \overline{1,11}$$

$$\sum_{j=1}^{11} y_{ij} \leq c_i \quad i = \overline{1,7}, j = \overline{1,11}$$

Поиск оптимального решения

Условие	Тип	П1	П2	П3	П4	П5	П6	П7	1 ч. работы, \$
Линия 1	3	0,5	0,7	0,8	0,4	0,7	0,8	0,5	20
Линия 2	3	0,65	0,91	1,04	-	0,91	1,04	0,63	15
Линия 3	2	0,35	0,49	0,56	0,28	0,49	0,56	0,35	30
Линия 4	2	0,25	0,35	0,4	-	0,35	0,4	0,25	40
Линия 5	1	-	0,28	0,32	0,16	0,28	0,32	0,2	50

	Б1	Б2	Б3	Б4	Б5	Б6	Б7	
Потребность	3500	1000	1000	2000	800	200	1000	шт.
Стоимость	25	26	28	24	27	29	23	\$

Затрачено		П1	П2	П3	П4	П5	П6	П7	Время работы	Стоимость работы	Загруженность линии
	Кол										
Линия 1	3	628	0	524	0	0	0	0	1152 <= 1152	23040,0	1
Линия 2	3	522	0	0	0	0	0	630	1152 <= 1152	17280,0	1
Линия 3	2	504	0	0	0	0	0	0	504 <= 768	15129,7	0,65667
Линия 4	2	0	350	138	0	200	80	0	768 <= 768	30720,0	1
Линия 5	1	0	0	0	320	64	0	0	384 <= 384	19200,0	1

Произведено	П1	П2	П3	П4	П5	П6	П7	
Линия 1	1256		655					
Линия 2	803						1000	
Линия 3	1441							
Линия 4		1000	345		571	200		
Линия 5				2000	229			
	Б1	Б2	Б3	Б4	Б5	Б6	Б7	
	3500	1000	1000	2000	800	200	1000	
	<=	<=	<=	<=	<=	<=	<=	
Потребность	3500	1000	1000	2000	800	200	1000	шт.
Прибыль	87500	26000	28000	48000	21600	5800	23000	

Целевая функция	L = 134530
-----------------	------------

рис. 1 excel таблица для расчёта оптимального решения

Параметры поиска решения

Оптимизировать целевую функцию:

\$O\$32

↑

До:

☒ Максимум

☐ Минимум

☐ Значения:

0

Изменяя ячейки переменных:

\$D\$15:\$J\$19

↑

В соответствии с ограничениями:

\$D\$29:\$J\$29 <= \$D\$31:\$J\$31
\$K\$15:\$K\$19 <= \$M\$15:\$M\$19

↑
↓

Добавить

Изменить

Удалить

Сбросить

Загрузить/сохранить

☒ Сделайте переменные без ограничений неотрицательными

Выберите метод решения:

Поиск решения лин. задач симплекс-методом

▼

Параметры

Метод решения

Для гладких нелинейных задач используйте поиск решения нелинейных задач методом ОПГ, для линейных задач - поиск решения линейных задач симплекс-методом, а для негладких задач - эволюционный поиск решения.

Справка

Найти решение

Закрыть

рис. 2 диалоговое окно «Параметры поиска решения»

Программная реализация

Языком для реализации был выбран kotlin, потому что на нём легко можно создать красивый пользовательский интерфейс.

Взаимодействие с файлами формата xlsx осуществлялось с помощью библиотеки Apache POI.

Пользовательский интерфейс реализован на Jetpack Compose.

Примеры вариантов использования приведены на изображениях 3 и 4.

ЛинииПродукты

Линия 1Линия 2Линия 3Линия 4Линия 5

Название
Линия 3

Количество линий
2

Время производства продукта 1
0.35

Время производства продукта 2
0.49

Время производства продукта 3
0.56

Время производства продукта 4
0.28

Время производства продукта 5
0.49

Время производства продукта 6
0.56

Время производства продукта 7
0.35

Цена
30.0

ЗагрузитьСохранить

ЛинииПродукты

Продукт	Ограничение	Стоимость
1	3500	25
2	100	26
3	10000	28
4	2	24
5	80	27
6	2000	29
7	10000	23

рис 3. и рис. 4 примеры использования программы

Используемые источники

<https://kotlinlang.org>

Kotlin Programming Language

<https://developer.android.com/jetpack/compose>

Jetpack Compose | Android Developers

Приложение А

ExcelReader

```
package Data
```

```
import manager.LineManager
import manager.ProductManager
import org.apache.poi.hssf.usermodel.HSSFWorkbook
import org.apache.poi.ss.usermodel.*
import org.apache.poi.ss.util.NumberToTextConverter
import org.apache.poi.xssf.usermodel.XSSFWorkbook
import repository.LineRepository
import repository.ProductRepository

import java.io.File
import java.io.FileInputStream
import java.util.ArrayList
import java.util.HashMap

object ExcelReader {
    // Чтение файла
    fun read(filename: String = "calc.xlsx") {
        val workbook = loadWorkbook(filename)
        val sheetIterator = workbook?.sheetIterator()
        if (sheetIterator != null) {
            while (sheetIterator.hasNext()) {
                val sheet: Sheet = sheetIterator.next()
                processSheet(sheet)
                println()
            }
        }
        else
            println("sheetIterator is null")
    }

    // Загрузка файла
    private fun loadWorkbook(filename: String): Workbook? {
        val extension = filename.substring(filename.lastIndexOf(".") + 1)
        val file = FileInputStream(File(filename))
        return when (extension) {
            "xls" -> HSSFWorkbook(file)
            "xlsx" -> XSSFWorkbook(file)
            else -> {
                println("Unknown Excel file extension: $extension")
                null
            }
        }
    }

    // Обработка страницы
    private fun processSheet(sheet: Sheet) {
        println("Sheet: " + sheet.getSheetName())
    }
}
```

```

val data = HashMap<Int, MutableList<Any>>()
val iterator = sheet.rowIterator()
var rowIndex = 0
while (iterator.hasNext()) {
    val row = iterator.next()
    processRow(data, rowIndex, row)
    rowIndex++
}
LineRepository.data.clear()
ProductRepository.data.clear()
for ((k,v) in data)
    if ( k <9 )
        LineRepository.add(LineManager.lineFormString(v))
    else if (k < 12 )
        ProductManager.productFromString(v)

println(ProductRepository.data.toString())
println(LineRepository.data.toString())
print("Sheet data:")
println(data)
}

// Обработка строки
private fun processRow(data: HashMap<Int, MutableList<Any>>, rowIndex: Int, row: Row) {
    data[rowIndex] = ArrayList()
    for (cell in row) {
        processCell(cell, data[rowIndex]!!)
    }
}

// Обработка ячейки
private fun processCell(cell: Cell, dataRow: MutableList<Any>) {
    when (cell.cellType) {
        CellType.STRING -> dataRow.add(cell.stringCellValue)
        CellType.NUMERIC -> if (DateUtil.isCellDateFormatted(cell)) {
            dataRow.add(cell.localDateTimeCellValue)
        } else {
            dataRow.add(NumberToTextConverter.toText(cell.numericCellValue))
        }
        CellType.BOOLEAN -> dataRow.add(cell.booleanCellValue)
        CellType.FORMULA -> dataRow.add(cell.cellFormula)
        else -> dataRow.add(" ")
    }
}
}

```

Line

```
package Data
```

```

data class Line(
    var name: String,
    var count: Int,
    var tp1: Double,
    var tp2: Double,
    var tp3: Double,
    var tp4: Double,
    var tp5: Double,

```

```
var tp6: Double,  
var tp7: Double,  
var cost: Double
```

Product

```
package Data
```

```
data class Product(  
    var limit: Int,  
    var cost: Int  
)
```

UpdateExcel

```
package Data
```

```
import org.apache.poi.ss.usermodel.Row  
import org.apache.poi.ss.usermodel.Sheet  
import org.apache.poi.xssf.usermodel.XSSFWorkbook  
import repository.LineRepository  
import repository.ProductRepository  
import java.io.File  
import java.io.FileInputStream  
import java.io.FileOutputStream  
import java.io.IOException  
import java.util.*
```

```
object UpdateExcel {
```

```
    fun changeCalls() {
```

```
        val file = FileInputStream(File("calc.xlsx"))  
        val workbook = XSSFWorkbook(file)  
        val sheet = workbook.getSheetAt(0)
```

```
        for (i in 0 until LineRepository.data.size) {  
            val header: Row = sheet.getRow(2+i)  
            var j = 1  
            val name = if ( "Линия".toRegex().containsMatchIn(LineRepository.data[i].name) ) {  
                LineRepository.data[i].name  
            } else {  
                "Линия ${LineRepository.data[i].name}"  
            }  
            header.getCell(j++).setCellValue(name)  
            header.getCell(j++).setCellValue(LineRepository.data[i].count.toString())  
            header.getCell(j++).setCellValue(LineRepository.data[i].tp1)  
            header.getCell(j++).setCellValue(LineRepository.data[i].tp2)  
            header.getCell(j++).setCellValue(LineRepository.data[i].tp3)  
            header.getCell(j++).setCellValue(LineRepository.data[i].tp4)  
            header.getCell(j++).setCellValue(LineRepository.data[i].tp5)  
            header.getCell(j++).setCellValue(LineRepository.data[i].tp6)  
            header.getCell(j++).setCellValue(LineRepository.data[i].tp7)  
            header.getCell(j).setCellValue(LineRepository.data[i].cost)  
        }  
    }
```

```

val productLimit: Row = sheet.getRow(9)
val productCost: Row = sheet.getRow(10)
for (i in 0 until ProductRepository.data.size ) {
    productLimit.getCell(3+i).setCellValue(ProductRepository.data[i].limit.toDouble())
    productCost.getCell(3+i).setCellValue(ProductRepository.data[i].cost.toDouble())
}

try {
    val out = FileOutputStream(File("calc.xlsx"))
    workbook.write(out)
    out.close()
    println("Значения успешно изменены")
} catch (e: IOException) {
    e.printStackTrace()
}
}

val checkedInputDigit: Double
get() {
    val digit = Scanner(System.`in`).nextDouble()
    if (digit < 0) {
        println("Значения не могут быть отрицательными. Повторите попытку! : ")
        checkedInputDigit
    }
    return digit
}
}

```

LineManager

```
package manager
```

```
import Data.Line
```

```

object LineManager {
    /**
     * Превращение строки в объект линии
     */
    fun lineFromString(inputList: MutableList<Any>) : Line? {
        if ( "Линия".toRegex().containsMatchIn(inputList.toString()) ) {
            val name = inputList[1].toString()
            val list = mutableListOf<Double>()
            for (i in 0..6)
                list.add( getDouble( inputList[3+i].toString() ) )
            val count = inputList[2].toString().substringAfter(" ").toInt()
            val cost = inputList[10].toString().toDouble()
            return Line(name, count, list[0],list[1],list[2],list[3],list[4],list[5],list[6], cost)
        }
        return null
    }

    private fun getDouble(str : String) =
        if ( str != "-" )
            str.toDouble()
        else

```



```
} 9999.0
```

ProductManager

```
package manager
```

```
import Data.Product
```

```
import repository.ProductRepository
```

```
object ProductManager {
```

```
    fun productFromString(inputList: MutableList<Any>) {
        if ( "Потребность".toRegex().containsMatchIn(inputList.toString()) ) {
            ProductRepository.data.clear()
            for (i in 0..6)
                ProductRepository.data.add(
                    Product(inputList[3 + i].toString().substringAfter(" ").toInt(), 0)
                )
        }
        else if ( "Стоимость".toRegex().containsMatchIn(inputList.toString()) )
            for (i in 0..6)
                ProductRepository.data[i].cost = inputList[3 + i].toString().toInt()
    }
}
```

LineRepository

```
package repository
```

```
import Data.Line
```

```
object LineRepository {
```

```
    val data = mutableListOf<Line>()
```

```
    fun add(line: Line?) =
        if (line != null) {
            if ( data.size == 5 )
                for (d in 0..4)
                    if (data[d].name == line.name)
                        data[d] = line
            data.add(line)
        }
        else
            false
}
```

ProductRepository

```
package repository
```

```
import Data.Product
```

```
object ProductRepository {
```

```
    val data = mutableListOf<Product>()
```

```
}
```

lineView

package screen

```
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.material.TextField
import androidx.compose.runtime.*
import repository.LineRepository
```

@Composable

```
fun lineView() {
    println("lineView")
    var viewLine by remember { mutableStateOf(0) }

    var name by remember { mutableStateOf(LineRepository.data[viewLine].name) }
    var count by remember { mutableStateOf(LineRepository.data[viewLine].count) }
    var tp1 by remember { mutableStateOf(LineRepository.data[viewLine].tp1) }
    var tp2 by remember { mutableStateOf(LineRepository.data[viewLine].tp2) }
    var tp3 by remember { mutableStateOf(LineRepository.data[viewLine].tp3) }
    var tp4 by remember { mutableStateOf(LineRepository.data[viewLine].tp4) }
    var tp5 by remember { mutableStateOf(LineRepository.data[viewLine].tp5) }
    var tp6 by remember { mutableStateOf(LineRepository.data[viewLine].tp6) }
    var tp7 by remember { mutableStateOf(LineRepository.data[viewLine].tp7) }
    var cost by remember { mutableStateOf(LineRepository.data[viewLine].cost) }
```

```
    Column {
        Row {
            for (i in 1..5) {
                Button(
                    onClick = {
                        println("onClick $i")
                        viewLine = i-1
                        name = LineRepository.data[viewLine].name
                        count = LineRepository.data[viewLine].count
                        tp1 = LineRepository.data[viewLine].tp1
                        tp2 = LineRepository.data[viewLine].tp2
                        tp3 = LineRepository.data[viewLine].tp3
                        tp4 = LineRepository.data[viewLine].tp4
                        tp5 = LineRepository.data[viewLine].tp5
                        tp6 = LineRepository.data[viewLine].tp6
                        tp7 = LineRepository.data[viewLine].tp7
                        cost = LineRepository.data[viewLine].cost
                    },
                    content = {
                        Text("Линия $i")
                    }
                )
            }
        }
    }
}
```

```
    Column {
        TextField(
            label = { Text("Название") },
```

```

        value = name,
        onChange = {
            name = it
            LineRepository.data[viewLine].name = it
        },
    )
    TextField(
        label = { Text("Количество линий") },
        value = count.toString(),
        onChange = {
            count = it.toInt()
            LineRepository.data[viewLine].count = it.toInt()
        }
    )
    TextField(
        label = { Text("Время производства продукта 1") },
        value = tp1.toString(),
        onChange = {
            tp1 = it.toDouble()
            LineRepository.data[viewLine].tp1 = it.toDouble()
        }
    )
    TextField(
        label = { Text("Время производства продукта 2") },
        value = tp2.toString(),
        onChange = {
            tp2 = it.toDouble()
            LineRepository.data[viewLine].tp2 = it.toDouble()
        }
    )
    TextField(
        label = { Text("Время производства продукта 3") },
        value = tp3.toString(),
        onChange = {
            tp3 = it.toDouble()
            LineRepository.data[viewLine].tp3 = it.toDouble()
        }
    )
    TextField(
        label = { Text("Время производства продукта 4") },
        value = tp4.toString(),
        onChange = {
            tp4 = it.toDouble()
            LineRepository.data[viewLine].tp4 = it.toDouble()
        }
    )
    TextField(
        label = { Text("Время производства продукта 5") },
        value = tp5.toString(),
        onChange = {
            tp5 = it.toDouble()
            LineRepository.data[viewLine].tp5 = it.toDouble()
        }
    )
    TextField(
        label = { Text("Время производства продукта 6") },
        value = tp6.toString(),
        onChange = {
            tp6 = it.toDouble()
            LineRepository.data[viewLine].tp6 = it.toDouble()
        }
    )

```

```

    )
    TextField(
        label = { Text("Время производства продукта 7") },
        value = tp7.toString(),
        onChange = { tp7 = it.toDouble()
            LineRepository.data[viewLine].tp7 = it.toDouble() }
    )
    TextField(
        label = { Text("Цена") },
        value = cost.toString(),
        onChange = {
            cost = it.toDouble()
            LineRepository.data[viewLine].cost = it.toDouble() }
    )
}
}
}
}

```

LineViewState

```
package screen
```

```
import androidx.compose.runtime.MutableState
```

```

data class LineViewState(
    val name : MutableState<String>,
    val count: MutableState<Int>,
    val tp: List<MutableState<Double>>,
    val cost : MutableState<Double>
)

```

productView

```
package screen
```

```

import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.text.BasicTextField
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.setValue
import androidx.compose.runtime.getValue
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import repository.ProductRepository

```

```

fun checkToDouble(string: String) =
    if ("".toRegex().containsMatchIn(string)) null
    else string

```

```
fun checkToInt(string: String) =
```

```

if (""[\\D]"".toRegex().containsMatchIn(string) ) null
else string

@Composable
fun productView() {
    println("productView")

    var l1 by remember { mutableStateOf(ProductRepository.data[0].limit) }
    var l2 by remember { mutableStateOf(ProductRepository.data[1].limit) }
    var l3 by remember { mutableStateOf(ProductRepository.data[2].limit) }
    var l4 by remember { mutableStateOf(ProductRepository.data[3].limit) }
    var l5 by remember { mutableStateOf(ProductRepository.data[4].limit) }
    var l6 by remember { mutableStateOf(ProductRepository.data[5].limit) }
    var l7 by remember { mutableStateOf(ProductRepository.data[6].limit) }

    var c1 by remember { mutableStateOf(ProductRepository.data[0].cost) }
    var c2 by remember { mutableStateOf(ProductRepository.data[1].cost) }
    var c3 by remember { mutableStateOf(ProductRepository.data[2].cost) }
    var c4 by remember { mutableStateOf(ProductRepository.data[3].cost) }
    var c5 by remember { mutableStateOf(ProductRepository.data[4].cost) }
    var c6 by remember { mutableStateOf(ProductRepository.data[5].cost) }
    var c7 by remember { mutableStateOf(ProductRepository.data[6].cost) }

    val tableData = (1..7).mapIndexed { index, item ->
        index to "Item $index"
    }
    val column1Weight = .3f
    val column2Weight = .3f
    val column3Weight = .4f
    LazyColumn(Modifier.fillMaxSize().padding(16.dp)) {
        // Here is the header
        item {
            Row(Modifier.background(Color.Gray)) {
                TableCell(text = "Продукт", weight = column1Weight)
                TableCell(text = "Ограничение", weight = column2Weight)
                TableCell(text = "Стоимость", weight = column3Weight)
            }
        }
        // Here are all the lines of your table.
        item {
            Row(Modifier.fillMaxWidth()) {
                TableCell(text = "1", weight = column1Weight)
                BasicTextField(
                    value = l1.toString(),
                    onValueChange = {
                        checkToInt(it)?.let { l1 = it.toInt() }
                        checkToInt(it)?.let { ProductRepository.data[0].limit = it.toInt() }
                    },
                    Modifier
                        .border(1.dp, Color.Black)
                        .weight(column2Weight)
                        .padding(8.dp)
                )
                BasicTextField(
                    value = c1.toString(),

```

```

        onValueChange = {
            checkToInt(it)?.let { c1 = it.toInt() }
            checkToInt(it)?.let { ProductRepository.data[0].cost = it.toInt() }
        },
        Modifier
            .border(1.dp, Color.Black)
            .weight(column3Weight)
            .padding(8.dp)
    )
}

item {
    Row(Modifier.fillMaxWidth()) {
        TableCell(text = "2", weight = column1Weight)
        BasicTextField(
            value = l2.toString(),
            onValueChange = {
                checkToInt(it)?.let { l2 = it.toInt() }
                checkToInt(it)?.let { ProductRepository.data[1].limit = it.toInt() }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column2Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = c2.toString(),
            onValueChange = {
                checkToInt(it)?.let { c2 = it.toInt() }
                checkToInt(it)?.let { ProductRepository.data[1].cost = it.toInt() }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column3Weight)
                .padding(8.dp)
        )
    }
}

item {
    Row(Modifier.fillMaxWidth()) {
        TableCell(text = "3", weight = column1Weight)
        BasicTextField(
            value = l3.toString(),
            onValueChange = {
                checkToInt(it)?.let { l3 = it.toInt() }
                checkToInt(it)?.let { ProductRepository.data[2].limit = it.toInt() }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column2Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = c3.toString(),

```

```

        onValueChange = {
            checkToInt(it)?.let { c3 = it.toInt() }
            checkToInt(it)?.let { ProductRepository.data[2].cost = it.toInt() }
        },
        Modifier
            .border(1.dp, Color.Black)
            .weight(column3Weight)
            .padding(8.dp)
    )
}

item {
    Row(Modifier.fillMaxWidth()) {
        TableCell(text = "4", weight = column1Weight)
        BasicTextField(
            value = l4.toString(),
            onValueChange = {
                checkToInt(it)?.let { l4 = it.toInt() }
                checkToInt(it)?.let { ProductRepository.data[3].limit = it.toInt() }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column2Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = c4.toString(),
            onValueChange = {
                checkToInt(it)?.let { c4 = it.toInt() }
                checkToInt(it)?.let { ProductRepository.data[3].cost = it.toInt() }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column3Weight)
                .padding(8.dp)
        )
    }
}

item {
    Row(Modifier.fillMaxWidth()) {
        TableCell(text = "5", weight = column1Weight)
        BasicTextField(
            value = l5.toString(),
            onValueChange = {
                checkToInt(it)?.let { l5 = it.toInt() }
                checkToInt(it)?.let { ProductRepository.data[4].limit = it.toInt() }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column2Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = c5.toString(),
            onValueChange = {
                checkToInt(it)?.let { c5 = it.toInt() }
            }
        )
    }
}

```



```

        checkToInt(it)?.let { ProductRepository.data[4].cost = it.toInt() }
    },
    Modifier
        .border(1.dp, Color.Black)
        .weight(column3Weight)
        .padding(8.dp)
    )
}
}

item {
    Row(Modifier.fillMaxWidth()) {
        TableCell(text = "6", weight = column1Weight)
        BasicTextField(
            value = l6.toString(),
            onChange = {
                checkToInt(it)?.let { l6 = it.toInt() }
                checkToInt(it)?.let { ProductRepository.data[5].limit = it.toInt() }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column2Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = c6.toString(),
            onChange = {
                checkToInt(it)?.let { c6 = it.toInt() }
                checkToInt(it)?.let { ProductRepository.data[5].cost = it.toInt() }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column3Weight)
                .padding(8.dp)
        )
    }
}

item {
    Row(Modifier.fillMaxWidth()) {
        TableCell(text = "7", weight = column1Weight)
        BasicTextField(
            value = l7.toString(),
            onChange = {
                checkToInt(it)?.let { l7 = it.toInt() }
                checkToInt(it)?.let { ProductRepository.data[6].limit = it.toInt() }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column2Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = c7.toString(),
            onChange = {
                checkToInt(it)?.let { c7 = it.toInt() }

```

```

        checkToInt(it)?.let { ProductRepository.data[6].cost = it.toInt() }
    },
    Modifier
        .border(1.dp, Color.Black)
        .weight(column3Weight)
        .padding(8.dp)
    )
}
}
}
}

@Composable
fun RowScope.TableCell(
    text: String,
    weight: Float
) {
    Text(
        text = text,
        Modifier
            .border(1.dp, Color.Black)
            .weight(weight)
            .padding(8.dp)
    )
}

```

App

```

import Data.ExcelReader
import Data.UpdateExcel
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import screen.lineView
import screen.productView

```

```

@Composable
fun App() {
    var vLine by remember { mutableStateOf(true) }

```

```

    Column {
        Row {
            Button(
                onClick = { vLine = true },
                content = { Text("Линии") }
            )
            Button(
                onClick = { vLine = false },
                content = { Text("Продукты") }
            )

```

```

    )
}

if (vLine) { lineView() }
else      { productView() }
downButton()
}
}

@Composable
fun downButton() {
    Row {
        Button(
            onClick = { ExcelReader.read() },
            content = { Text("Загрузить") }
        )
        Button(
            onClick = { UpdateExcel.changeCalls() },
            content = { Text("Сохранить") }
        )
    }
}
}

```

Main

```

import Data.ExcelReader
import androidx.compose.ui.window.Window
import androidx.compose.ui.window.application

fun main() = application {

    //Загрузка при запуске
    ExcelReader.read()

    Window(onCloseRequest = ::exitApplication) {
        App()
    }
}

```