Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САРАТОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

**Деревья**

КУРСОВАЯ РАБОТА

студента 3 курса 331 группы

специальности 10.05.01 «Компьютерная безопасность»

факультета компьютерных наук и информационных технологий

Подкидышева Ивана Александровича

Научный руководитель

профессор, к.ф.-м.н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Н. Салий

подпись, дата

Зав. кафедрой

профессор, к.ф.-м.н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Н. Салий

подпись дата

Саратов 2017 г.

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 3

1 Основные алгебраические конструкции 4

2 Деревья 7

3 Код Прюфера 10

4 Практическая часть 14

4.1 Построение плоского изображения дерева 14

4.2 Построение кода Прюфера 15

4.3 Построение плоского изображения дерева из кода Прюфера 16

ЗАКЛЮЧЕНИЕ 20

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 23

ПРИЛОЖЕНИЯ 24

**ВВЕДЕНИЕ**

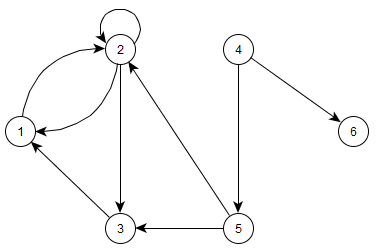
Графы (ориентированные и неориентированные) являются одной из важнейших структур в программировании. Большой пласт практических задач сводится к графам, например, прокладка маршрута на карте или начертание схем приборов (топология межсоединений элементов на печатной плате или электросхеме представляет собой граф или гиперграф).

Дерево как часть теории графов также является важной структурой. Многие алгоритмы принимают на вход дерево или строят деревья, обладающие некоторыми специальными свойствами. Например, дерево кратчайших путей строится обходом в ширину графа. В реальной жизни деревья, а именно их плоские изображения, являются очень наглядным способом представления различных планов, задач.

Если говорить о коде Прюфера, то он является крайне эффективным способом задания дерева. Код состоит всего лишь из элементов, где n – количество вершин дерева. Таким образом для передачи, например, по сети дерева как объекта, достаточно будет передать лишь его код Прюфера – обычный массив, а не сложные структуры, списки или матрицы.

**1 Основные алгебраические конструкции**

Под *ориентированным графом* (*орграфом*) понимается пара , где V – конечное непустое множество (*вершины* графа), а – отношение на множестве V (пара (*u*, *v*) α называется *дугой* орграфа с *началом* *u* и *концом v*). Отношение α называется отношением *смежности*, а соответствующую ему двоичную булеву матрицу – *матрицей смежности* орграфа G.

Каждому *орграфу* можно сопоставить некоторое изображение – диаграмму, на которой вершины орграфа представляются точками (или кружками), а дуги – непрерывными кривыми (например, прямолинейными отрезками), ориентированными стрелкой от начальной вершины к конечной. Ясно, что один и тот же орграф допускает много различных изображений. Пример матрицы смежности и соответствующего ей изображения орграфа представлен на рисунке 1.

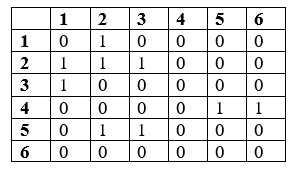


Рисунок 1 – Пример орграфа

*Неориентированным орграфом* (*графом*) называется пара , где α – симметричное и антирефлексивное отношение на множестве вершин V. Из каждого орграфа можно получить неориентированный граф , называемый *симметризацией* *орграфа* . Чтобы построить изображение симметризации G, в изображении орграфа убирают стрелки на дугах и петли, а каждую пару встречных дуг (*u*, *v*) и (*v*, *u*) заменяют одной линией. Различные орграфы могут иметь совпадение симметризации. Дуги неориентированного графа обычно называют *рёбрами*. Ребро, образованное вершинами *u* и *v*, иногда обозначают через {*u*, *v*}. Пример симметризации изображён на рисунке 2 (симметризация орграфа на рисунке 1).

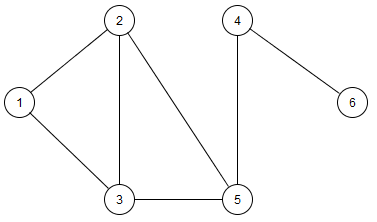


Рисунок 2 – Пример графа

Как видим у всех дуг теперь нет направления, пара встречных дуг (1, 2) и (2, 1) орграфа на рисунке 1 были заменены на одно ребро (1, 2), а петля (2, 2) была удалена.

Пусть – некоторый орграф, *v* *V* – одна из его вершин. *Степенью исхода* вершины v называется число дуг орграфа , имеющих своим началом v, т.е. . *Степень захода* вершины *v* – это число дуг, имеющих *v* своим концом, т.е. . В неориентированном графе . Число называется *степенью вершины v*. Если равно единице, то такая веершина называется *висячей* или *листом*. В графе на рисунке 2: степень вершины 1 равна 2, степень вершины 6 равна 1 (висячая вершина).

*Путём* в графе называется последовательность рёбер, в которой каждые два соседних ребра имеют общую вершину и никакое ребро не встречается более одного раза. Путь в графе удобно задавать перечислением входящих в него вершин в порядке их прохождения. Если начальная и конечная вершина совпадают, путь называется *циклическим*. Путь, каждая вершина которого принадлежит не более чем двум его рёбрам называется *простым*. Простой циклический путь называется *циклом*. Если простой путь не является циклом, то его называют *цепью*, а первую и последнюю вершины пути – *концами* цепи. На графе на рисунке 2 можно выделить простой циклический путь *2*, *3*, *5*, *2*; циклический путь *5*, *3*, *2*, *1*, *3*, *5*; цепь *3*, *2*, *5*, *4*, *6*, где *3* и *2* – концы цепи [1].

**2 Деревья**

Изображение графа *G* называется *плоским*, если никакие два ребра в этом изображении не пересекаются. Графы, имеющие плоское изображение, называются *планарными*.

Важным классом планарных графов являются деревья. *Дерево* – это связный граф, в котором нет циклов.

Теорема 1: граф *G* тогда и только тогда является деревом, когда в нём выполняется одно из следующих условий:

1) любые две вершины графа *G* соединены единственной цепью;

2) *G* – связный граф и число вершин в нём на единицу больше числа рёбер: *n* = *m* + 1;

3) *G* не содержит циклов и *n* = *m* + 1.

Рассмотрим пример дерева, изображённый на рисунке 3. В нём 8 вершин и 7 рёбер, нет циклов, а каждые две различные вершины соединены единственной цепью.

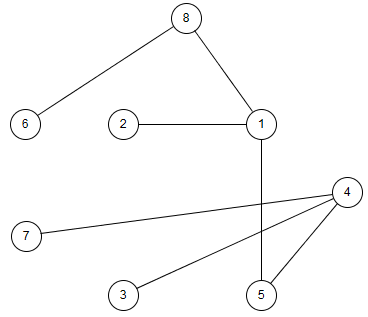


Рисунок 3 – Пример дерева

Дерево с одной вершиной называется *тривиальным*. Из теоремы 1 следует, что в каждом нетривиальном дереве существуют по крайней мере две висячие вершины. Это утверждение следует из того, что сумма степеней всех вершин *n*-вершинного дерева равна 2*n* – 2. Если же в дереве нет висячих вершин, то , а при одной висячей вершине было бы .

Плоское изображение произвольного дерева можно получить следующим образом. Произвольно выберем некоторую вершину v0 (*корень дерева*). Если *k* – расстояние от до наиболее удалённых от корня вершин, проведём *k* + 1 горизонталей, нумеруя их снизу-вверх числами 0, 1, … *k*. На нулевой горизонтали поместим вершину , а на горизонтали 1 расположим произвольным образом вершины, смежные с (т.е. находящиеся на расстоянии 1 от корня). Пусть это будут (слева направо) вершины . На горизонтали 2 разместим вершины, удалённые на расстояние 2 от корня , таким образом, что сначала будут идти вершины, смежные с , за ними – вершины, смежные с и т.д. Завершив в соответствии с этим правилом размещение всех вершин на горизонталях, соединим каждую из них прямолинейными отрезками со смежными с ней вершинами следующего уровня (если таковые есть).

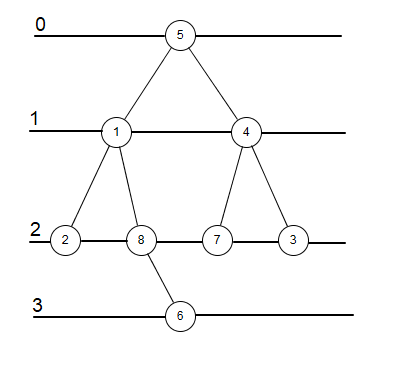


Рисунок 4 – Построение планарного изображения дерева

Выбрав в качестве корня вершину с номером 5 графа на рисунке 3, построим по этому алгоритму плоское изображение этого дерева. На 1-ом уровне расположатся вершины с номерами 1 и 4; на 2-ом: 2, 8, 7, 3; на 3-ем: 6. Полученное изображение показано на рисунке 4 [1].

**3 Код Прюфера**

Рассмотрим произвольное дерево с n заданными и пронумерованными в произвольном порядке вершинами. Спрашивается: сколько существует таких деревьев? Английский математик А. Кэли предположил, что деревьев с *n* пронумерованными вершинами ровно столько, сколько можно образовать последовательностей вида длины *n* – 2, элементы которых выбираются из элементов множества , то есть . Один и тот же элемент в такой последовательности может встречаться по нескольку раз. Покажем, что таких последовательностей .

Немецкий математик Прюфер указал алгоритм, следуя которому каждому дереву можно поставить во взаимно однозначное соответствие такую последовательность длины *n* – 2, элементы которой черпаются из множества.

Алгоритм: рассмотрим, например, дерево на рисунке 5. Выберем у него висячую вершину с наименьшим номером. Это вершина 2. Удалим её вместе с принадлежащим ей ребром. Запишем в ответ 1. Единица – номер вершины полученного дерева, ближайшей к удалённой. Такая вершина всего одна, так как удалённая вершина была висячей, следовательно, она имеет всего одну смежную вершину – вершину 1.

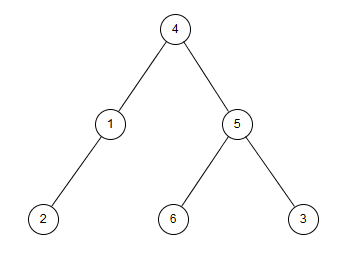


Рисунок 5

Выполним эту же операцию повторно. Вновь выберем висячую вершину с наименьшим номером. Это вершина 1. Удаляем её вместе с ребром. Записываем в ответ 4 (четыре – номер вершины, ближайшей к вершине 1). Второй шаг завершен. На данный момент ответ имеет вид (1, 4) (рисунок 6).

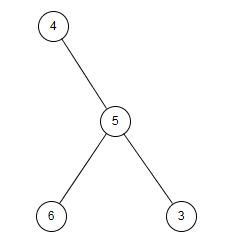


Рисунок 6

Повторяя эту процедуру до тех пор, пока не останутся две висячие вершины (в данном случае это будут вершины 5 и 6), получаем для данного дерева определённую единственным образом последовательность (*1*, *4*, *5*, *5*) длины *n* – 2.

Обратный алгоритм: рассмотрим последовательность длины n – 2, составленную из элементов множества , например, ту же последовательность (*1*, *4*, *5*, *5*). Будем, следуя методу Прюфера, соединять соответствующие вершины рёбрами и получать деревья.

Рассмотрим первый элемент последовательности – 1. Найдём вершину с наименьшим номером, не использованную ранее и не встречающуюся далее в последовательности. Это вершина с номером 2. Добавим в итоговое дерево ребро (4, 2). Пометим число 2 как использованное. Перейдем к следующему шагу алгоритма. Рассмотрим второй элемент последовательности – 1. Найдем наименьшее число – номер вершины, неиспользованный ранее и не встречающийся далее в коде Прюфера – число 5. Пометим число 5 как использованное, а в итоговое дерево добавим ребро (1, 5). Граф на этом шаге изображён на рисунке 7.

Будем подобным образом продолжать наполнять дерево рёбрами, пока не исчерпаем всю последовательность.

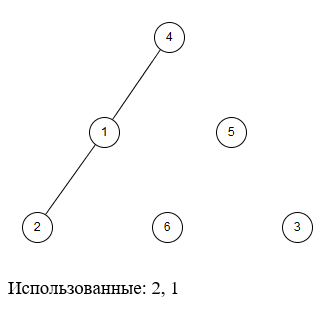


Рисунок 7

Осталось добавить одно ребро – ребро между теми двумя висячими вершинами, оставшимися после составления последовательности Прюфера. Один из концов этого ребра – вершина с номером, равным последнему элементу последовательности – вершина с номером 5. Другой конец ребра – вершина с наименьшим неиспользованным ранее номером – вершина с номером 6.

Исходя из алгоритмов, можно выделить два основных свойства последовательности Прюфера:

* По окончании построения кода Прюфера в дереве останутся неудалёнными две вершины. Одной из них точно будет вершина с максимальным номером – , а вот про другую вершину ничего определённого сказать нельзя.
* Каждая вершина встречается в коде Прюфера определённое число раз, равное её степени минус один. Это легко понять, если заметить, что вершина удаляется из дерева в момент, когда её степень равна единице — т.е. к этому моменту все смежные с ней рёбра, кроме одного, были удалены. Для двух оставшихся после построения кода вершин это утверждение тоже верно.

Таким образом по последовательности длины *n* – 2, состоящей из элементов множества *M*, мы однозначным образом получили дерево. Причем данное и конечное деревья совпадают. Следовательно, можно говорить о взаимно однозначном соответствии между множеством всех деревьев с вершинами, пронумерованным с 1 по *n*, и множеством всех последовательностей из элементов множества длины *n* – 2. Убеждаемся, что различных деревьев с пронумерованными вершинами [2].

**4 Практическая часть**

В качестве практической части было предложено написать программу, моделирующую процесс построения дерева, построения кода Прюфера, а также построения дерева по данному коду Прюфера. Программа была написана на языке Object Pascal в программной среде Embarcadero RAD Studio 10.1 Berlin с использованием фреймворка FMX. Полный код программы представлен в приложении 1, но мы рассмотрим её основные фрагменты. Фактическое построение изображения не будет подробно рассматриваться, так как эта задача в целом возложена на фреймворк и не относится к основной теме данной работы.

**4.1 Построение плоского изображения дерева**

Исходный граф, по которому строится дерево, задается матрицей смежности. Изображение дерева в программе имеет вид упорядоченного списка вершин – объектов класса *TVertex*. В объекте класса содержится информация о номере вершины, её уровне в дереве, ссылка на вершину-предка (null, если это корень дерева) и список ссылок на дочерние вершины (у висячих вершин этот список будет пуст). Вершина *v* имеет позицию *i* в списке дерева, если *v* стоит на

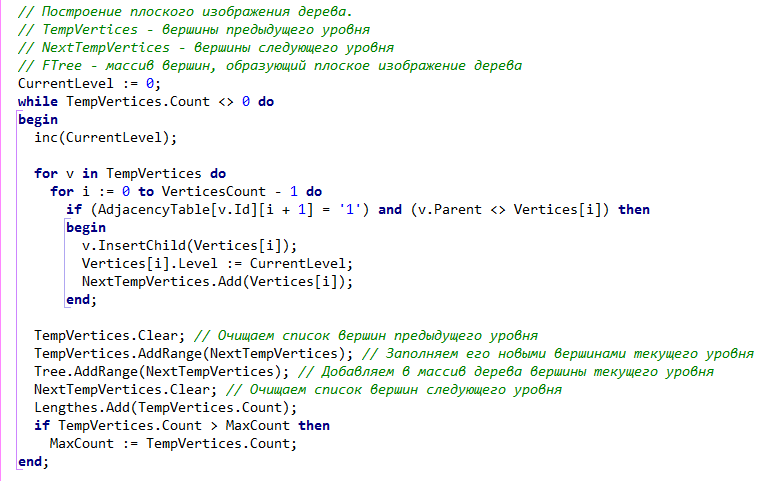


Рисунок 8

уровне *k* дерева и , где – число вершин на *j*-ом уровне дерева, – порядковый номер вершины *v* на *k*-ом уровне. Код построения этого списка представлен на рисунке 8. Этот список вместе со служебной информацией представлен классом *TTree*.

**4.2 Построение кода Прюфера**

Код Прюфера в программе представляет из себя простой массив из *n* – 2 элементов, состоящий из элементов множества *M*. Помимо самого кода для визуализации также потребовалось поддерживать упорядоченный список удалённых рёбер в порядке их удаления из графа. Алгоритм принимает на вход матрицу смежности графа, на выходе – код Прюфера и упорядоченный список удалённых рёбер. Код алгоритм представлен на рисунке 9.

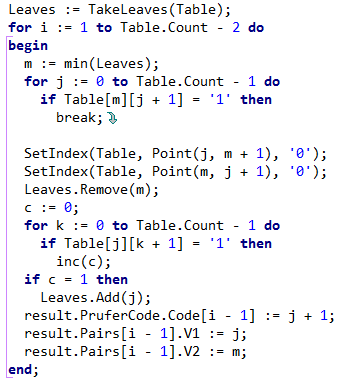


Рисунок 9

Таким образом программа повторяет алгоритм, описанный в главе 2. Запускаем цикл *n* – 2 раз, на каждом его шаге программа находит минимальный лист, ребро, связанное с этим листом, удаляет это ребро из графа, удаляет висячую вершину из списка всех висящих (список *Leaves*) и добавляет, если необходимо, смежную с удалённой вершину (добавление происходит, если эта вершина стала висячей). При этом в ответ добавляется номер этой смежной вершины.

**4.3 Построение плоского изображения дерева из кода Прюфера**

Алгоритм построения плоского изображения дерева из кода Прюфера принимает на вход последовательность Прюфера, а на выход отдает упорядоченный список вершин, как в главе 3.1 – объект класса *TTree*. Программа повторяет алгоритм, описанный в главе 2. Фрагмент кода представлен на рисунке 10.

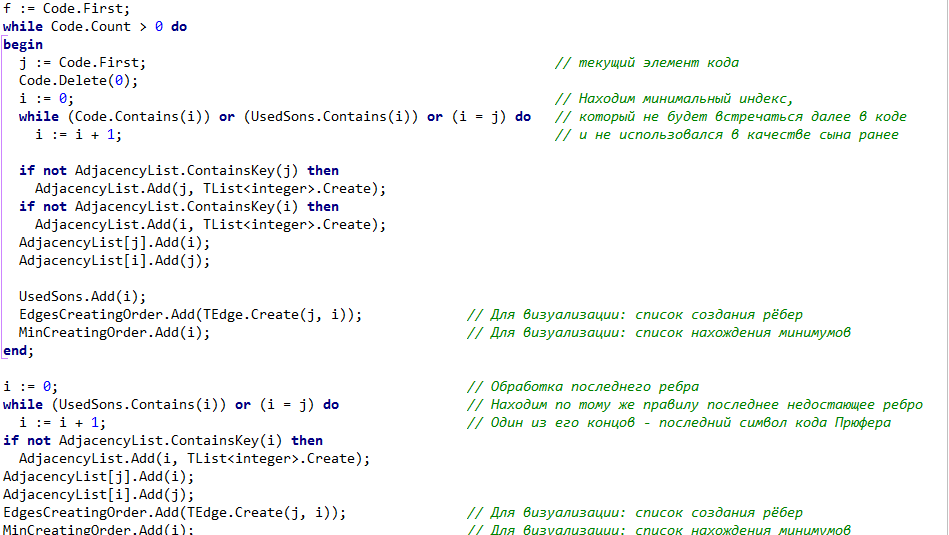


Рисунок 10

Основной цикл выполняется пока в коде Прюфера ещё есть номера вершин. На каждом шаге: вынимаем первый номер из последовательности – *j*, удаляем первый (только что вытащенный) элемент из последовательности. Затем ищем минимальный номер вершины, не встречающийся в коде и не использованный ранее (массив *UsedSons*) – переменная *i*. Добавляем в граф ребро (*i*, *j*), обновляем служебные списки. Граф для удобства будет представлен списком смежности – *AdjacencyList* – список списков. *i*-ый список содержит номера вершин, смежных с вершиной *i*.

**ЗАКЛЮЧЕНИЕ**

Построение плоского изображения дерева является достаточно простой задачей, в то время как построение кода Прюфера и построение дерева по коду Прюфера являются достаточно серьезными задачами. Однако самое применение кода Прюфера для кодирования графа даёт большой выигрыш в объёме, занимаемым деревом в памяти компьютера. С другой же стороны, код Прюфера не позволяется анализировать исходное дерево, модифицировать и вообще каким-либо способом изменять его в конкретных целях.

В заключение можно сказать, что код Прюфера является эффективным контейнером, хранящим в себе исходное дерево. Помимо этого, алгоритм конструирования кода Прюфера даёт точную оценку количества неориентированных деревьев, состоящих из *n* вершин, а именно .

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Богомолов, А.М. Алгебраические основы теории дискретных систем [Текст] / А.М. Богомолов, В.Н. Салий –Москва: «Физико-математическая литература» РАН, 1997. –367с.

2. Березина, Л.Ю. Графы и их применение [Текст] / Березина Л.Ю. –Москва «Просвещение», 1979. –142с.

**ПРИЛОЖЕНИЕ 1**

Файл Tree.dpr:

program Tree;

uses

System.StartUpCopy,

FMX.Forms,

MainUnit in 'MainUnit.pas' {MainForm},

GraphUnit in 'GraphUnit.pas',

VisualizationUnit in 'VisualizationUnit.pas',

PruferVisualizationUnit in 'PruferVisualizationUnit.pas',

PruferCodeUnit in 'PruferCodeUnit.pas',

VisualizationUtils in 'VisualizationUtils.pas';

{$R \*.res}

begin

Application.Initialize;

Application.CreateForm(TMainForm, MainForm);

Application.Run;

end.

Файл MainUnit.fmx:

object MainForm: TMainForm

Left = 0

Top = 0

Caption = 'TreeViewer'

ClientHeight = 729

ClientWidth = 1023

WindowState = wsMaximized

FormFactor.Width = 320

FormFactor.Height = 480

FormFactor.Devices = [Desktop]

OnCloseQuery = FormCloseQuery

OnDestroy = FormDestroy

OnShow = FormShow

DesignerMasterStyle = 0

object ControlLayout: TLayout

Align = Left

Size.Width = 377.000000000000000000

Size.Height = 729.000000000000000000

Size.PlatformDefault = False

TabOrder = 1

object ControlUsualLayout: TLayout

Align = Top

Size.Width = 377.000000000000000000

Size.Height = 409.000000000000000000

Size.PlatformDefault = False

TabOrder = 0

object BuildButton: TButton

StyledSettings = [Family, Style, FontColor]

Position.X = 144.000000000000000000

Position.Y = 264.000000000000000000

Size.Width = 121.000000000000000000

Size.Height = 22.000000000000000000

Size.PlatformDefault = False

TabOrder = 0

Text = #1055#1086#1089#1090#1088#1086#1080#1090#1100

TextSettings.Font.Size = 16.000000000000000000

OnClick = BuildButtonClick

end

object RootEdit: TEdit

Touch.InteractiveGestures = [LongTap, DoubleTap]

TabOrder = 1

FilterChar = '0123456789'

Text = '1'

TextSettings.Font.Size = 16.000000000000000000

Position.X = 24.000000000000000000

Position.Y = 264.000000000000000000

Size.Width = 105.000000000000000000

Size.Height = 22.000000000000000000

Size.PlatformDefault = False

StyledSettings = [Family, Style, FontColor]

end

object RootLabel: TLabel

StyledSettings = [Family, Style, FontColor]

Position.X = 24.000000000000000000

Position.Y = 240.000000000000000000

Size.Width = 209.000000000000000000

Size.Height = 17.000000000000000000

Size.PlatformDefault = False

TextSettings.Font.Size = 16.000000000000000000

Text = #1053#1086#1084#1077#1088' '#1074#1077#1088#1096#1080#1085#1099' '#1082#1086#1088#1085#1103

end

object TableLabel: TLabel

StyledSettings = [Family, Style, FontColor]

Position.X = 24.000000000000000000

Position.Y = 8.000000000000000000

Size.Width = 209.000000000000000000

Size.Height = 17.000000000000000000

Size.PlatformDefault = False

TextSettings.Font.Size = 16.000000000000000000

Text = #1052#1072#1090#1088#1080#1094#1072' '#1089#1084#1077#1078#1085#1086#1089#1090#1080

end

object TableMemo: TMemo

Touch.InteractiveGestures = [Pan, LongTap, DoubleTap]

DataDetectorTypes = []

Lines.Strings = (

'010011'

'101000'

'010100'

'001000'

'100000'

'100000')

StyledSettings = [Style, FontColor]

TextSettings.Font.Family = 'Consolas'

TextSettings.Font.Size = 16.000000000000000000

Position.X = 24.000000000000000000

Position.Y = 32.000000000000000000

Size.Width = 209.000000000000000000

Size.Height = 201.000000000000000000

Size.PlatformDefault = False

TabOrder = 4

Viewport.Width = 205.000000000000000000

Viewport.Height = 197.000000000000000000

end

object PruferCodeBuildLabel: TLabel

StyledSettings = [Family, Style, FontColor]

Position.X = 24.000000000000000000

Position.Y = 312.000000000000000000

Size.Width = 121.000000000000000000

Size.Height = 25.000000000000000000

Size.PlatformDefault = False

TextSettings.Font.Size = 16.000000000000000000

Text = #1050#1086#1076' '#1055#1088#1102#1092#1077#1088#1072':'

end

object PruferCodeBuiltLabel: TLabel

StyledSettings = [Family]

Position.X = 24.000000000000000000

Position.Y = 352.000000000000000000

Size.Width = 209.000000000000000000

Size.Height = 25.000000000000000000

Size.PlatformDefault = False

TextSettings.Font.Size = 16.000000000000000000

TextSettings.Font.StyleExt = {00060000000000000004000000}

TextSettings.FontColor = xFF1604AF

Text = '< '#1085#1077' '#1087#1086#1089#1090#1088#1086#1077#1085' >'

end

object ToPruferBuildButton: TButton

StyledSettings = [Family, Style, FontColor]

Position.X = 144.000000000000000000

Position.Y = 312.000000000000000000

Size.Width = 121.000000000000000000

Size.Height = 22.000000000000000000

Size.PlatformDefault = False

TabOrder = 7

Text = #1055#1086#1089#1090#1088#1086#1080#1090#1100' '#1082#1086#1076

TextSettings.Font.Size = 16.000000000000000000

OnClick = ToPruferBuildButtonClick

end

object ComboBox1: TComboBox

Position.X = 248.000000000000000000

Position.Y = 32.000000000000000000

Size.Width = 81.000000000000000000

Size.Height = 22.000000000000000000

Size.PlatformDefault = False

TabOrder = 8

OnChange = ComboBox1Change

end

object DownButton: TButton

StyledSettings = [Family, Style, FontColor]

Position.X = 24.000000000000000000

Position.Y = 384.000000000000000000

TabOrder = 9

Text = #1042#1085#1080#1079

TextSettings.Font.Size = 14.000000000000000000

OnClick = DownButtonClick

end

end

object ControlPruferLayout: TLayout

Align = Client

Anchors = [akLeft, akTop]

Size.Width = 377.000000000000000000

Size.Height = 320.000000000000000000

Size.PlatformDefault = False

TabOrder = 1

object PruferCodeEdit: TEdit

Touch.InteractiveGestures = [LongTap, DoubleTap]

TabOrder = 1

FilterChar = '0123456789,'

Text = '3,2,1,1'

TextSettings.Font.Size = 16.000000000000000000

Position.X = 24.000000000000000000

Position.Y = 40.000000000000000000

Size.Width = 233.000000000000000000

Size.Height = 22.000000000000000000

Size.PlatformDefault = False

StyledSettings = [Family, Style, FontColor]

end

object PruferCodeLabel: TLabel

StyledSettings = [Family, Style, FontColor]

Position.X = 24.000000000000000000

Position.Y = 16.000000000000000000

Size.Width = 145.000000000000000000

Size.Height = 17.000000000000000000

Size.PlatformDefault = False

TextSettings.Font.Size = 16.000000000000000000

Text = #1050#1086#1076' '#1055#1088#1102#1092#1077#1088#1072

end

object PruferBuildButton: TButton

StyledSettings = [Family, Style, FontColor]

Position.X = 24.000000000000000000

Position.Y = 80.000000000000000000

TabOrder = 4

Text = #1055#1086#1089#1090#1088#1086#1080#1090#1100

TextSettings.Font.Size = 16.000000000000000000

OnClick = PruferBuildButtonClick

end

object AnimationSpeedLabel: TLabel

Anchors = [akLeft, akBottom]

StyledSettings = [Family, Style, FontColor]

Position.X = 16.000000000000000000

Position.Y = 248.000000000000000000

Size.Width = 129.000000000000000000

Size.Height = 17.000000000000000000

Size.PlatformDefault = False

TextSettings.Font.Size = 14.000000000000000000

Text = #1057#1082#1086#1088#1086#1089#1090#1100' '#1072#1085#1080#1084#1072#1094#1080#1080

end

object TrackBar1: TTrackBar

Anchors = [akLeft, akBottom]

CanParentFocus = True

Orientation = Horizontal

Position.X = 16.000000000000000000

Position.Y = 280.000000000000000000

Size.Width = 129.000000000000000000

Size.Height = 19.000000000000000000

Size.PlatformDefault = False

TabOrder = 6

Value = 50.000000000000000000

OnTracking = TrackBar1Tracking

end

end

end

object GraphsLayout: TLayout

Align = Client

Size.Width = 646.000000000000000000

Size.Height = 729.000000000000000000

Size.PlatformDefault = False

TabOrder = 3

object TreeImage: TImage

MultiResBitmap = <

item

end>

Align = Client

Size.Width = 309.000000000000000000

Size.Height = 729.000000000000000000

Size.PlatformDefault = False

end

object GraphsLeftLayout: TLayout

Align = Left

Size.Width = 337.000000000000000000

Size.Height = 729.000000000000000000

Size.PlatformDefault = False

TabOrder = 0

object GraphImage: TImage

MultiResBitmap = <

item

end>

Align = Top

Size.Width = 337.000000000000000000

Size.Height = 337.000000000000000000

Size.PlatformDefault = False

end

object PruferGraphLayout: TImage

MultiResBitmap = <

item

end>

Align = Client

Size.Width = 337.000000000000000000

Size.Height = 392.000000000000000000

Size.PlatformDefault = False

end

object PruferGraphImage: TImage

MultiResBitmap = <

item

end>

Align = Client

Size.Width = 337.000000000000000000

Size.Height = 392.000000000000000000

Size.PlatformDefault = False

end

end

object ClearButton: TButton

Anchors = [akTop, akRight]

Position.X = 560.000000000000000000

Position.Y = 8.000000000000000000

TabOrder = 1

Text = #1054#1095#1080#1089#1090#1080#1090#1100

OnClick = ClearButtonClick

end

end

end

Файл MainUnit.pas:

unit MainUnit;

interface

uses

// Delphi system includings

System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants, System.Generics.Collections, System.IOUtils,

// FMX includings

FMX.Types, FMX.Controls, FMX.Forms, FMX.Graphics, FMX.Dialogs, FMX.Layouts, FMX.Objects, FMX.StdCtrls, FMX.Controls.Presentation,

FMX.ScrollBox, FMX.Memo, FMX.Edit, FMX.DialogService,

// Project includings

GraphUnit, VisualizationUnit, PruferCodeUnit, PruferVisualizationUnit, VisualizationUtils, FMX.ListBox;

type

TMainForm = class(TForm)

ControlLayout: TLayout;

TableMemo: TMemo;

BuildButton: TButton;

RootEdit: TEdit;

TableLabel: TLabel;

RootLabel: TLabel;

GraphsLayout: TLayout;

GraphImage: TImage;

TreeImage: TImage;

GraphsLeftLayout: TLayout;

PruferGraphLayout: TImage;

ControlUsualLayout: TLayout;

ControlPruferLayout: TLayout;

PruferCodeEdit: TEdit;

PruferCodeLabel: TLabel;

PruferBuildButton: TButton;

PruferGraphImage: TImage;

ClearButton: TButton;

PruferCodeBuildLabel: TLabel;

PruferCodeBuiltLabel: TLabel;

ToPruferBuildButton: TButton;

AnimationSpeedLabel: TLabel;

TrackBar1: TTrackBar;

ComboBox1: TComboBox;

DownButton: TButton;

procedure BuildButtonClick(Sender: TObject);

procedure PruferBuildButtonClick(Sender: TObject);

procedure FormDestroy(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure ToPruferBuildButtonClick(Sender: TObject);

procedure ClearButtonClick(Sender: TObject);

procedure TrackBar1Tracking(Sender: TObject);

procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);

procedure ComboBox1Change(Sender: TObject);

procedure DownButtonClick(Sender: TObject);

private

{ Private declarations }

public

Tree: TTree;

PruferTree: TPruferTree;

end;

var

MainForm: TMainForm;

ToPruferCode: TPruferCode;

TreeComponents: TCompleteComponents;

implementation

{$R \*.fmx}

procedure InitBox;

var

s, NormalS: string;

Files: TStringDynArray;

begin

Files := TDirectory.GetFiles(ExtractFilePath(ParamStr(0)) + '\Examples');

for s in Files do

begin

NormalS := s.Substring(s.LastIndexOf('\') + 1);

if NormalS.Substring(0, 2) = 'ex' then

MainForm.ComboBox1.Items.Add(NormalS.Substring(0, NormalS.IndexOf('.')));

end;

end;

procedure ErrorMessage(const &Message: string);

begin

TDialogService.MessageDialog(&Message, TMsgDlgType.mtError, [TMsgDlgBtn.mbOK], TMsgDlgBtn.mbOK, 0, nil);

end;

function PruferFromVisual(CodeEdit: TEdit): TPruferCode;

var

i: integer;

Elements: TArray<string>;

begin

Elements := CodeEdit.Text.Split([',']);

SetLength(result.Code, Length(Elements));

for i := 0 to Length(Elements) - 1 do

result.Code[i] := Elements[i].ToInteger - 1;

end;

procedure TMainForm.FormShow(Sender: TObject);

begin

ControlUsualLayout.Height := ControlLayout.Height \* 0.6;

GraphsLeftLayout.Width := (GraphsLeftLayout.Width + TreeImage.Width) \* 0.3;

GraphImage.Height := GraphsLeftLayout.Height \* 0.5;

Tree := nil;

PruferTree := nil;

TreeComponents := nil;

InitBox;

end;

procedure TMainForm.FormCloseQuery(Sender: TObject; var CanClose: Boolean);

begin

CanClose := true;

if BUILDING then

CanClose := false;

end;

procedure TMainForm.FormDestroy(Sender: TObject);

begin

if Tree <> nil then

Tree.Destroy;

if PruferTree <> nil then

PruferTree.Destroy;

if TreeComponents <> nil then

TreeComponents.Destroy;

end;

procedure TMainForm.BuildButtonClick(Sender: TObject);

begin

if Tree <> nil then

Tree.Destroy;

if TreeComponents <> nil then

TreeComponents.Destroy;

if not ValidateTable(TableMemo.Lines) then

begin

ErrorMessage('Неправильно введена таблица смежности!');

exit;

end;

Tree := TTree.Create(TableMemo.Lines, StrToInt(RootEdit.Text) - 1);

TreeComponents := Visualize(Tree, GraphImage, TreeImage, BuildButton);

end;

procedure TMainForm.ToPruferBuildButtonClick(Sender: TObject);

var

PruferVisual: TPruferVisual;

begin

if Tree = nil then

begin

ErrorMessage('Сначала постройте дерево!');

exit;

end;

PruferVisual := ToPrufer(TableMemo.Lines);

ToPruferCode := PruferVisual.PruferCode;

VisualizeToPrufer(PruferVisual, Tree, TreeComponents, PruferCodeBuiltLabel);

end;

procedure TMainForm.TrackBar1Tracking(Sender: TObject);

begin

DELAY := MIN\_DELAY + Trunc((MAX\_DELAY - MIN\_DELAY) \* (1 - TrackBar1.Value / 100));

end;

procedure TMainForm.PruferBuildButtonClick(Sender: TObject);

begin

if PruferTree <> nil then

PruferTree.Destroy;

if not ValidatePruferCode(PruferCodeEdit.Text) then

begin

ErrorMessage('Код введён некоррректно!');

exit;

end;

PruferTree := TPruferTree.Create(PruferFromVisual(PruferCodeEdit));

VisualizeFromPrufer(PruferTree, PruferGraphImage, TreeImage);

end;

procedure TMainForm.ClearButtonClick(Sender: TObject);

begin

if Tree <> nil then

Tree.Destroy;

Tree := nil;

ClearImage(TreeImage);

end;

procedure TMainForm.ComboBox1Change(Sender: TObject);

var

t: TTree;

begin

if Tree <> nil then

Tree.Destroy;

Tree := nil;

TableMemo.Lines.Clear;

PruferCodeBuiltLabel.Text := '< не построен >';

TableMemo.Lines.LoadFromFile(ExtractFilePath(ParamStr(0))+'\Examples\'+ComboBox1.Items[ComboBox1.ItemIndex]+'.txt');

t := TTree.Create(TableMemo.Lines, 0);

VisualizeGraph(t, GraphImage).Destroy;

t.Destroy;

ClearImage(TreeImage);

end;

procedure TMainForm.DownButtonClick(Sender: TObject);

begin

PruferCodeEdit.Text := PruferCodeBuiltLabel.Text;

end;

end.

Файл GraphUnit.pas:

unit GraphUnit;

interface

uses

// Delphi system includings

System.Classes, System.Generics.Collections, System.SysUtils, System.Types;

type

TEdge = record

Id1, Id2: integer;

constructor Create(Id1, Id2: integer);

end;

TVertex = class

public

Id: integer;

Level: integer;

Parent: TVertex;

Children: TArray<TVertex>;

private

procedure InsertChild(Vertex: Tvertex);

constructor Create(Id: integer);

end;

TVertexList = TList<TVertex>;

TVertexArray = array of TVertex;

TTree = class

public

TreeHeight: integer;

VerticesCount: integer;

RootVertex: TVertex;

MaxCount: integer;

// containers

Vertices: TList<TVertex>;

Tree: TList<TVertex>;

Edges: TList<TEdge>;

Lengthes: TList<integer>;

function IndexOfVertex(Index: integer): integer;

constructor Create(AdjacencyTable: TStrings; RootIndex: integer);

destructor Destroy; override;

end;

function ValidateTable(Table: TStrings): boolean;

implementation

function ValidateTable(Table: TStrings): boolean;

var

i, j: integer;

begin

result := false;

for i := 0 to Table.Count - 1 do

if Table[i].Length <> Table.Count then

exit;

for i := 0 to Table.Count - 1 do

for j := 0 to i do

if (i = j) and (Table[i][j + 1] = '1') then

exit

else if Table[i][j + 1] <> Table[j][i + 1] then

exit;

result := true;

end;

{ TVertex }

constructor TVertex.Create(Id: integer);

begin

Self.Id := Id;

Level := 0;

Parent := nil;

end;

procedure TVertex.InsertChild(Vertex: Tvertex);

var

v: TVertex;

begin

for v in Children do

if v.Id = Vertex.Id then

raise Exception.Create('Inserting child exception');

Vertex.Parent := Self;

SetLength(Children, Length(Children) + 1);

Children[Length(Children) - 1] := Vertex;

end;

{ TEdge }

constructor TEdge.Create(Id1, Id2: integer);

begin

Self.Id1 := Id1;

Self.Id2 := Id2;

end;

{ TTreeRecord }

constructor TTree.Create(AdjacencyTable: TStrings; RootIndex: integer);

var

TempVertices, NextTempVertices: TList<TVertex>;

CurrentLevel, i, j: integer;

v: TVertex;

begin

Vertices := TList<TVertex>.Create;

Tree := TList<TVertex>.Create;

Edges := TList<TEdge>.Create;

Lengthes := TList<integer>.Create;

VerticesCount := AdjacencyTable.Count;

for i := 0 to AdjacencyTable.Count - 1 do

Vertices.Add(TVertex.Create(i));

// Заполнение списка рёбер. Пробег по верхней левой половине матрицы смежности

for i := 0 to AdjacencyTable.Count - 1 do

for j := 1 to i + 1 do

if AdjacencyTable[i][j] = '1' then

Edges.Add(TEdge.Create(i, j - 1));

RootVertex := Vertices[RootIndex];

Tree.Add(RootVertex);

Lengthes.Add(1);

MaxCount := 1;

TempVertices := TList<TVertex>.Create;

NextTempVertices := TList<TVertex>.Create;

TempVertices.Add(RootVertex);

// Построение плоского изображения дерева.

// TempVertices - вершины предыдущего уровня

// NextTempVertices - вершины следующего уровня

// FTree - массив вершин, образующий плоское изображение дерева

CurrentLevel := 0;

while TempVertices.Count <> 0 do

begin

inc(CurrentLevel);

for v in TempVertices do

for i := 0 to VerticesCount - 1 do

if (AdjacencyTable[v.Id][i + 1] = '1') and (v.Parent <> Vertices[i]) then

begin

v.InsertChild(Vertices[i]);

Vertices[i].Level := CurrentLevel;

NextTempVertices.Add(Vertices[i]);

end;

TempVertices.Clear; // Очищаем список вершин предыдущего уровня

TempVertices.AddRange(NextTempVertices); // Заполняем его новыми вершинами текущего уровня

Tree.AddRange(NextTempVertices); // Добавляем в массив дерева вершины текущего уровня

NextTempVertices.Clear; // Очищаем список вершин следующего уровня

Lengthes.Add(TempVertices.Count);

if TempVertices.Count > MaxCount then

MaxCount := TempVertices.Count;

end;

TempVertices.Destroy;

NextTempVertices.Destroy;

TreeHeight := CurrentLevel;

end;

destructor TTree.Destroy;

var

v: TVertex;

begin

for v in Vertices do

v.Destroy;

Vertices.Destroy;

Tree.Destroy;

Edges.Destroy;

Lengthes.Destroy;

end;

function TTree.IndexOfVertex(Index: integer): integer;

var

i: integer;

begin

for i := 0 to Tree.Count - 1 do

if Tree[i].Id = Index then

begin

result := i;

exit;

end;

raise Exception.Create('Vertex ' + Index.ToString + ' didn''t found');

end;

end.

Файл PruferCodeUnit.pas:

unit PruferCodeUnit;

interface

uses

// Delphi system includings

System.Classes, System.Generics.Collections, System.Types, System.Math, System.SysUtils,

// Project includings

GraphUnit;

type

TPruferCode = record

Code: TArray<integer>;

function ToString: string;

end;

TPruferTree = class(TTree)

type

TPruferTreeVerticesPair = record

ParentVertex: TVertex;

SonVertex: TVertex;

constructor Create(Vertex1, Vertex2: TVertex);

end;

public

PruferCode: TPruferCode;

FirstVertex: integer;

UsedSons: TList<integer>;

EdgesCreatingOrder: TList<TEdge>;

MinCreatingOrder: TList<integer>;

function GetOrderedVertex(Index: integer): TPruferTreeVerticesPair;

constructor Create(PruferCode: TPruferCode);

destructor Destroy; override;

end;

TPruferTreeVerticesPair = TPruferTree.TPruferTreeVerticesPair;

TPruferPair = record

V1, V2: integer;

end;

TPruferVisual = record

PruferCode: TPruferCode;

Pairs: TArray<TPruferPair>;

end;

function ToPrufer(AdjacencyTable: TStrings): TPruferVisual;

function ValidatePruferCode(Code: string): boolean;

implementation

function ValidatePruferCode(Code: string): boolean;

var

Vertex: integer;

s: string;

Vertices: TArray<string>;

begin

result := false;

Code.Replace(' ', '');

Vertices := Code.Split([',']);

if Length(Vertices) < 1 then

exit;

for s in Vertices do

if not TryStrToInt(s, Vertex) then

exit;

result := true;

end;

function TakeTableFromList(AdjacencyList: TDictionary<integer, TList<integer>>): TStringList;

var

s: string;

i, j, m: integer;

Keys: TArray<integer>;

begin

result := TStringList.Create;

Keys := AdjacencyList.Keys.ToArray;

m := -1;

for i in Keys do

if i > m then

m := i;

inc(m);

for i := 0 to m - 1 do

begin

s := '';

for j := 0 to m - 1 do

if AdjacencyList[i].Contains(j) then

s := s + '1'

else

s := s + '0';

result.Add(s);

end;

end;

function CopyStrings(Strings: TStrings): TStringList;

var

i: integer;

begin

result := TStringList.Create;

for i := 0 to Strings.Count - 1 do

result.Add(Strings[i]);

end;

function TakeLeaves(AdjacencyTable: TStringList): TList<integer>;

var

i, j: integer;

c: char;

begin

result := TList<integer>.Create;

for i := 0 to AdjacencyTable.Count - 1 do

begin

j := 0;

for c in AdjacencyTable[i] do

if c = '1' then

inc(j);

if j = 1 then

result.Add(i);

end;

end;

function min(l: TList<integer>): integer;

var

i: integer;

begin

result := l[0];

for i := 1 to l.Count - 1 do

if l[i] < result then

result := l[i];

end;

procedure SetIndex(Table: TStrings; Coord: TPoint; Value: char);

var

s: string;

begin

s := Table[Coord.X];

s[Coord.Y] := Value;

Table[Coord.X] := s;

end;

function ToPrufer(AdjacencyTable: TStrings): TPruferVisual;

var

i, j, m, k, c: integer;

Leaves: TList<integer>;

Table: TStringList;

begin

Table := CopyStrings(AdjacencyTable);

SetLength(result.PruferCode.Code, Table.Count - 2);

SetLength(result.Pairs, Table.Count - 2);

Leaves := TakeLeaves(Table);

for i := 1 to Table.Count - 2 do

begin

m := min(Leaves);

for j := 0 to Table.Count - 1 do

if Table[m][j + 1] = '1' then

break;

SetIndex(Table, Point(j, m + 1), '0');

SetIndex(Table, Point(m, j + 1), '0');

Leaves.Remove(m);

c := 0;

for k := 0 to Table.Count - 1 do

if Table[j][k + 1] = '1' then

inc(c);

if c = 1 then

Leaves.Add(j);

result.PruferCode.Code[i - 1] := j + 1;

result.Pairs[i - 1].V1 := j;

result.Pairs[i - 1].V2 := m;

end;

Leaves.Destroy;

Table.Destroy;

end;

function check(arr: TArray<integer>; index, k: integer): boolean;

var

i: integer;

begin

result := true;

for i := index to Length(arr) - 1 do

if arr[i] = k then

begin

result := false;

exit;

end;

end;

{ TPruferCode }

function TPruferCode.ToString: string;

var

i: integer;

begin

for i := 0 to Length(Code) - 1 do

begin

result := result + Code[i].ToString;

if i = Length(Code) - 1 then

result := result + ' '

else

result := result + ', ';

end;

end;

{ TPruferTree }

constructor TPruferTree.Create(PruferCode: TPruferCode);

var

i, j, f: integer;

Code: TList<integer>;

AdjacencyTable: TStringList;

AdjacencyList: TDictionary<integer, TList<integer>>;

Keys: TArray<integer>;

begin

Self.PruferCode := PruferCode;

FirstVertex := PruferCode.Code[0] - 1;

Code := TList<integer>.Create;

Code.AddRange(PruferCode.Code);

UsedSons := TList<integer>.Create;

AdjacencyList := TDictionary<integer, TList<integer>>.Create;

EdgesCreatingOrder := TList<TEdge>.Create;

MinCreatingOrder := TList<integer>.Create;

f := Code.First;

while Code.Count > 0 do

begin

j := Code.First; // текущий элемент кода

Code.Delete(0);

i := 0; // Находим минимальный индекс,

while (Code.Contains(i)) or (UsedSons.Contains(i)) or (i = j) do // который не будет встречаться далее в коде

i := i + 1; // и не использовался в качестве сына ранее

if not AdjacencyList.ContainsKey(j) then

AdjacencyList.Add(j, TList<integer>.Create);

if not AdjacencyList.ContainsKey(i) then

AdjacencyList.Add(i, TList<integer>.Create);

AdjacencyList[j].Add(i);

AdjacencyList[i].Add(j);

UsedSons.Add(i);

EdgesCreatingOrder.Add(TEdge.Create(j, i)); // Для визуализации: список создания рёбер

MinCreatingOrder.Add(i); // Для визуализации: список нахождения минимумов

end;

i := 0; // Обработка последнего ребра

while (UsedSons.Contains(i)) or (i = j) do // Находим по тому же правилу последнее недостающее ребро

i := i + 1; // Один из его концов - последний символ кода Прюфера

if not AdjacencyList.ContainsKey(i) then

AdjacencyList.Add(i, TList<integer>.Create);

AdjacencyList[j].Add(i);

AdjacencyList[i].Add(j);

EdgesCreatingOrder.Add(TEdge.Create(j, i)); // Для визуализации: список создания рёбер

MinCreatingOrder.Add(i); // Для визуализации: список нахождения минимумов

AdjacencyTable := TakeTableFromList(AdjacencyList);

inherited Create(AdjacencyTable, f);

AdjacencyTable.Destroy;

Keys := AdjacencyList.Keys.ToArray;

for i in Keys do

AdjacencyList[i].Destroy;

AdjacencyList.Destroy;

end;

destructor TPruferTree.Destroy;

begin

UsedSons.Destroy;

EdgesCreatingOrder.Destroy;

MinCreatingOrder.Destroy;

inherited;

end;

function TPruferTree.GetOrderedVertex(Index: integer): TPruferTreeVerticesPair;

begin

result := TPruferTreeVerticesPair.Create(Vertices[EdgesCreatingOrder[Index].Id1], Vertices[EdgesCreatingOrder[Index].Id2]);

end;

{ TPruferTree.TPruferTreeVerticesPair }

constructor TPruferTree.TPruferTreeVerticesPair.Create(Vertex1, Vertex2: TVertex);

begin

if Vertex1.Parent = Vertex2 then

begin

ParentVertex := Vertex2;

SonVertex := Vertex1;

end

else

begin

ParentVertex := Vertex1;

SonVertex := Vertex2;

end;

end;

end.

Файл VisualizationUtils.pas:

unit VisualizationUtils;

interface

uses

// Delphi system includings

System.Generics.Collections, System.UITypes, System.Types, System.Math, System.SysUtils, System.Threading, System.Classes,

// FMX includings

FMX.Graphics, FMX.Objects, FMX.Types, FMX.Controls, FMX.StdCtrls,

// Project includings

GraphUnit, PruferCodeUnit;

const

MAX\_DELAY: integer = 2000;

MIN\_DELAY: integer = 100;

type

TComponents = class

Circles: TList<TCircle>;

Points: TList<TPointF>;

Lines: TDictionary<string, TLine>;

constructor Create;

destructor Destroy; override;

end;

TTreeCoefficient = record

X: single;

Y: single;

Other: single;

constructor Create(X: single; Y: single; Other: single);

end;

function GetCoefficientForTree(Image: TImage; Tree: TTree): TTreeCoefficient;

function PutHorizontalLines(Image: TImage; Tree: TTree; Coefficient: TTreeCoefficient): TList<TLine>;

procedure RemoveHorizontalLines(Lines: TList<TLine>);

procedure PutGraphCircles(Image: TImage; Count: integer; Coefficient: single; Result: TComponents);

procedure PutTreeCircles(Image: TImage; Tree: TTree; Coefficient: TTreeCoefficient; Result: TComponents);

function ConnectVertices(Image: TImage; VertexFather, VertexSon: TVertex; Points: TList<TPointF>; Coefficient: TTreeCoefficient): TLine;

function ConnectGraphVertices(Image: TImage; Tree: TTree; Edge: TEdge; Points: TList<TPointF>; Coefficient: single): TLine;

procedure ClearImage(TreeImage: TImage);

var

DELAY: integer;

BUILDING: boolean;

implementation

function AngleOfVectors(const P1, P2: TPointF): Double;

begin

Result := RadToDeg(arccos((P1.x \* P2.X + P1.Y \* P2.Y) / P1.Length / P2.Length));

end;

procedure Swap(var P1, P2: TPointF);

var

p3: TPointF;

begin

p3 := p1;

p1 := p2;

p2 := p3;

end;

function GetCoefficientForTree(Image: TImage; Tree: TTree): TTreeCoefficient;

var

kx, ky: single;

begin

ky := Image.Height / (Tree.TreeHeight + 1);

kx := Min(Image.Width / (Tree.MaxCount + 1) \* 0.6, ky); // 0.6 - some random coefficient

result := TTreeCoefficient.Create(kx, ky, Min(Image.Width, Image.Height) / 7);

end;

function PutHorizontalLines(Image: TImage; Tree: TTree; Coefficient: TTreeCoefficient): TList<TLine>;

var

i: integer;

l: TLine;

begin

result := TList<TLine>.Create;

for i := 0 to Tree.TreeHeight - 1 do

begin

TThread.Synchronize(nil,

procedure

begin

l := TLine.Create(Image);

l.Parent := Image;

l.LineType := TLineType.Top;

l.Width := Image.Width \* 0.8;

l.Position.Point := TPointF.Create(Image.Width \* 0.1, Image.Height / 2 - Tree.TreeHeight \* Coefficient.Y / 2

+ (i + 0.25) \* Coefficient.Y);

l.Stroke.Thickness := 2;

l.Stroke.Color := TAlphaColorRec.Grey;

with TText.Create(l) do

begin

Parent := l;

Text := 'Ур = ' + (i + 1).ToString;

Height := Coefficient.Other / 6;

Width := Height \* Text.Length;

Font.Size := Height \* 0.8;

Position.Point := TPointF.Create(0, -Height \* 1.1);

VertTextAlign := TTextAlign.Leading;

HorzTextAlign := TTextAlign.Leading;

end;

end);

result.Add(l);

sleep(DELAY);

end;

end;

procedure RemoveHorizontalLines(Lines: TList<TLine>);

var

l: TLine;

begin

for l in Lines do

begin

TThread.Synchronize(nil,

procedure

begin

l.Destroy;

end);

sleep(DELAY);

end;

Lines.Destroy;

end;

procedure PutGraphCircles(Image: TImage; Count: integer; Coefficient: single; Result: TComponents);

var

c: TCircle;

i: integer;

begin

for i := 0 to Count - 1 do

begin

c := TCircle.Create(Image);

with c do

begin

Parent := Image;

result.Points.Add(TPointF.Create(Image.Width / 2, Image.Height / 2) +

Coefficient \* TPointF.Create(cos(2 \* PI \* i / Count), sin(2 \* PI \* i / Count)));

Position.Point := result.Points.Last;

Fill.Color := TAlphaColorRec.Black;

Width := Coefficient \* 0.15;

Height := Coefficient \* 0.15;

end;

with TText.Create(c) do

begin

Parent := c;

if c.Position.Point.X < Image.Width \* 0.5 then

Position.Point := PointF(-Coefficient \* 0.15, 0)

else

Position.Point := PointF(c.Width, 0);

Width := Coefficient \* 0.2;

Height := Coefficient \* 0.1;

Font.Size := Coefficient \* 0.1;

Text := (i + 1).ToString;

HorzTextAlign := TTextAlign.Center;

VertTextAlign := TTextAlign.Center;

end;

result.Circles.Add(c);

end;

end;

procedure PutTreeCircles(Image: TImage; Tree: TTree; Coefficient: TTreeCoefficient; Result: TComponents);

var

v: TVertex;

i, TempLevel: integer;

p: TPointF;

c: TCircle;

Points: TArray<TPointF>;

begin

SetLength(Points, Tree.VerticesCount);

i := 0;

TempLevel := 0;

for v in Tree.Tree do

begin

if v.Level <> TempLevel then

begin

inc(TempLevel);

i := 0;

sleep(DELAY);

end;

TThread.Synchronize(nil,

procedure

begin

c := TCircle.Create(Image);

with c do

begin

Parent := Image;

Points[v.Id] := PointF(Image.Width / 2 - Tree.Lengthes[v.Level] \* Coefficient.X / 2 + i \* Coefficient.X + Coefficient.X / 2,

Image.Height / 2 - Tree.TreeHeight \* Coefficient.Y / 2 + (v.Level + 0.25) \* Coefficient.Y - 0.125 \* Coefficient.X);

Position.Point := Points[v.Id];

Width := Coefficient.X / 4;

Height := Coefficient.X / 4;

Fill.Color := TAlphaColorRec.Black;

end;

with TText.Create(c) do

begin

Parent := c;

Font.Size := Coefficient.X / 8;

if c.Position.Point.X <= Image.Width / 2 then

Position.Point := PointF(-Coefficient.X / 8, -Coefficient.X / 32)

else

Position.Point := PointF(Coefficient.X \* 0.285, -Coefficient.X / 32);

Text := (v.Id + 1).ToString;

VertTextAlign := TTextAlign.Leading;

HorzTextAlign := TTextAlign.Leading;

end;

end);

Result.Circles.Add(c);

inc(i);

end;

for p in Points do

result.Points.Add(p);

end;

function ConnectVertices(Image: TImage; VertexFather, VertexSon: TVertex; Points: TList<TPointF>; Coefficient: TTreeCoefficient): TLine;

var

l: TLine;

begin

TThread.Synchronize(nil,

procedure

begin

l := TLine.Create(Image);

with l do

begin

TagString := VertexFather.Id.ToString + ',' + VertexSon.Id.ToString;

Parent := Image;

LineType := TLineType.Diagonal;

Position.Point := Points[VertexFather.Id] + PointF(Coefficient.X / 8, Coefficient.X / 8);

Width := abs(Points[VertexSon.Id].X - Points[VertexFather.Id].X);

Height := Points[VertexSon.Id].Y - Points[VertexFather.Id].Y;

RotationCenter.Point := TPointF.Zero;

if Points[VertexSon.Id].X - Points[VertexFather.Id].X < 0 then

RotationAngle := AngleOfVectors(PointF(Width, Height), (PointF(-Width, Height)));

if Width = 0 then

begin

LineType := TLineType.Left;

Width := Coefficient.X;

end;

Fill.Color := TAlphaColorRec.Grey;

Stroke.Thickness := 3;

Stroke.Color := TAlphaColorRec.Grey;

SendToBack;

end;

end);

result := l;

end;

function ConnectGraphVertices(Image: TImage; Tree: TTree; Edge: TEdge; Points: TList<TPointF>; Coefficient: single): TLine;

var

l: TLine;

begin

TThread.Synchronize(nil,

procedure

var

p1, p2, p3: TPointF;

vp: TPruferTreeVerticesPair;

begin

l := TLine.Create(Image);

with l do

begin

vp := TPruferTreeVerticesPair.Create(Tree.Vertices[Edge.Id1], Tree.Vertices[Edge.Id2]);

if Tree.Vertices[Edge.Id1].Parent = Tree.Vertices[Edge.Id2] then

TagString := Edge.Id2.ToString + ',' + Edge.Id1.ToString

else

TagString := Edge.Id1.ToString + ',' + Edge.Id2.ToString;

p1 := Points[Edge.Id1];

p2 := Points[Edge.Id2];

if p1.Y > p2.Y then

begin

p3 := p1;

p1 := p2;

p2 := p3;

end;

Parent := Image;

LineType := TLineType.Top;

Position.Point := p1 + PointF(Coefficient \* 0.075, Coefficient \* 0.075);

Width := (p2 - p1).Length;

Height := Coefficient \* 0.1;

RotationCenter.Point := TPointF.Zero;

RotationAngle := AngleOfVectors(PointF(1, 0), p2 - p1);

Fill.Color := TAlphaColorRec.Grey;

Stroke.Thickness := 3;

Stroke.Color := TAlphaColorRec.Grey;

SendToBack;

end;

end);

result := l;

end;

procedure ClearImage(TreeImage: TImage);

var

k: integer;

begin

for k := TreeImage.ChildrenCount - 1 downto 0 do

TreeImage.Children[k].Destroy;

end;

{ TTreeCoefficients }

constructor TTreeCoefficient.Create(X, Y, Other: single);

begin

Self.X := X;

Self.Y := Y;

Self.Other := Other;

end;

{ TGraphResult }

constructor TComponents.Create;

begin

Circles := TList<TCircle>.Create;

Points := TList<TPointF>.Create;

Lines := TDictionary<string, TLine>.Create;

end;

destructor TComponents.Destroy;

begin

Circles.Destroy;

Points.Destroy;

Lines.Destroy;

inherited;

end;

initialization

DELAY := MIN\_DELAY + Trunc((MAX\_DELAY - MIN\_DELAY) / 2);

BUILDING := false;

end.

Файл VisualizationUnit.pas:

unit VisualizationUnit;

interface

uses

// Delphi system includings

System.Generics.Collections, System.UITypes, System.Types, System.Math, System.SysUtils, System.Threading, System.Classes,

// FMX includings

FMX.Graphics, FMX.Objects, FMX.Types, FMX.Controls, FMX.StdCtrls,

// Project includings

GraphUnit, VisualizationUtils;

type

TCompleteComponents = class

Graph: TComponents;

Tree: TComponents;

constructor Create(Graph: TComponents; Tree: TComponents);

destructor Destroy; override;

end;

function VisualizeTree(Tree: TTree; Image: TImage; Button: TButton): TComponents;

function VisualizeGraph(Tree: TTree; Image: TImage): TComponents;

function Visualize(Tree: TTree; GraphImage, TreeImage: TImage; Button: TButton): TCompleteComponents;

implementation

function VisualizeGraph(Tree: TTree; Image: TImage): TComponents;

var

i: integer;

l: TLine;

Coefficient: single;

begin

for i := Image.ChildrenCount - 1 downto 0 do

Image.Children[i].Destroy;

result := TComponents.Create;

Coefficient := Min(Image.Width \* 0.875, Image.Height \* 0.875) / 2;

PutGraphCircles(Image, Tree.VerticesCount, Coefficient, result);

for i := 0 to Tree.Edges.Count - 1 do

begin

l := ConnectGraphVertices(Image, Tree, Tree.Edges[i], result.Points, Coefficient);

result.Lines.Add(l.TagString, l);

end;

end;

function VisualizeTree(Tree: TTree; Image: TImage; Button: TButton): TComponents;

var

z: integer;

TreeResult: TComponents;

begin

BUILDING := true;

for z := Image.ChildrenCount - 1 downto 0 do

Image.Children[z].Destroy;

Button.Enabled := false;

TreeResult := TComponents.Create;

TTask.Run(

procedure

var

l: TLine;

V, ChildV: TVertex;

Lines: TList<TLine>;

Coefficient: TTreeCoefficient;

begin

// Coefficient

Coefficient := GetCoefficientForTree(Image, Tree);

// Lines

Lines := PutHorizontalLines(Image, Tree, Coefficient);

// Circles

PutTreeCircles(Image, Tree, Coefficient, TreeResult);

// Connecting Vertices

Sleep(DELAY);

for V in Tree.Tree do

begin

for ChildV in V.Children do

begin

l := ConnectVertices(Image, V, ChildV, TreeResult.Points, Coefficient);

TreeResult.Lines.Add(l.TagString, l);

end;

if Length(V.Children) <> 0 then

sleep(DELAY);

end;

RemoveHorizontalLines(Lines);

sleep(DELAY);

TThread.Synchronize(nil,

procedure

begin

with TText.Create(Image) do

begin

Parent := Image;

Text := 'Высота дерева - ' + Tree.TreeHeight.ToString;

Height := Coefficient.Other / 4;

Font.Size := Height \* 0.8;

Width := Height \* Text.Length;

VertTextAlign := TTextAlign.Center;

HorzTextAlign := TTextAlign.Leading;

Position.Point := TPointF.Create(Height \* 1.2, Height \* 0.5);

end;

Button.Enabled := true;

end);

BUILDING := false;

end);

result := TreeResult;

end;

function Visualize(Tree: TTree; GraphImage, TreeImage: TImage; Button: TButton): TCompleteComponents;

var

G, T: TComponents;

begin

G := VisualizeGraph(Tree, GraphImage);

T := VisualizeTree(Tree, TreeImage, Button);

result := TCompleteComponents.Create(G, T);

end;

{ TTreeCompleteResult }

constructor TCompleteComponents.Create(Graph, Tree: TComponents);

begin

Self.Graph := Graph;

Self.Tree := Tree;

end;

destructor TCompleteComponents.Destroy;

begin

Graph.Destroy;

Tree.Destroy;

inherited;

end;

end.

Файл PruferVisualizationUnit.pas:

unit PruferVisualizationUnit;

interface

uses

// Delphy system includings

System.Math, System.Types, System.Threading, System.SysUtils, System.Classes, System.Generics.Collections, System.UITypes,

// FMX includings

FMX.Graphics, FMX.Objects, FMX.StdCtrls, FMX.Types,

// Project includings

GraphUnit, PruferCodeUnit, VisualizationUnit, VisualizationUtils;

type

TTexts = record

MinText, UsedText: TText;

Coefficient: TTreeCoefficient;

constructor Create(Image: TImage; Coefficient: TTreeCoefficient);

procedure Destroy;

end;

procedure VisualizeFromPrufer(PruferTree: TPruferTree; GraphImage, TreeImage: TImage);

procedure VisualizeToPrufer(PruferVisual: TPruferVisual; Tree: TTree; GT: TCompleteComponents; L: TLabel);

implementation

function PutTexts(Image: TImage; Coefficient: TTreeCoefficient): TTexts;

var

t: TTexts;

begin

TThread.Synchronize(nil,

procedure

begin

t := TTexts.Create(Image, Coefficient);

end);

result := t;

end;

procedure SetMinimum(Texts: TTexts; NewMinimum: integer);

begin

TThread.Synchronize(nil,

procedure

begin

Texts.MinText.Text := 'Минимальный - ' + NewMinimum.ToString;

end);

end;

procedure AddUsed(Texts: TTexts; NewUsed: integer);

begin

TThread.Synchronize(nil,

procedure

begin

with Texts do

begin

if UsedText.Text[UsedText.Text.Length - 1] <> ':' then

UsedText.Text := UsedText.Text + ', ';

UsedText.Text := UsedText.Text + NewUsed.ToString;

end;

end);

end;

procedure VisualizeFromPrufer(PruferTree: TPruferTree; GraphImage, TreeImage: TImage);

var

Coefficient: single;

TreeCoefficient: TTreeCoefficient;

GraphVisualComponents, TreeVisualComponents: TComponents;

begin

BUILDING := true;

ClearImage(TreeImage);

ClearImage(GraphImage);

Coefficient := Min(GraphImage.Width, GraphImage.Height) \* 0.875 / 2;

TreeCoefficient := GetCoefficientForTree(TreeImage, PruferTree);

GraphVisualComponents := TComponents.Create;

TreeVisualComponents := TComponents.Create;

PutGraphCircles(GraphImage, PruferTree.VerticesCount, Coefficient, GraphVisualComponents);

TTask.Run(

procedure

var

i: integer;

VerticesPair: TPruferTreeVerticesPair;

Texts: TTexts;

begin

PutTreeCircles(TreeImage, PruferTree, TreeCoefficient, TreeVisualComponents);

Texts := PutTexts(TreeImage, TreeCoefficient);

for i := 0 to PruferTree.EdgesCreatingOrder.Count - 1 do

begin

VerticesPair := PruferTree.GetOrderedVertex(i);

SetMinimum(Texts, PruferTree.MinCreatingOrder[Min(i, PruferTree.MinCreatingOrder.Count - 1)] + 1);

sleep(DELAY);

ConnectVertices(TreeImage, VerticesPair.ParentVertex, VerticesPair.SonVertex, TreeVisualComponents.Points,

TreeCoefficient);

ConnectGraphVertices(GraphImage, PruferTree, PruferTree.EdgesCreatingOrder[i], GraphVisualComponents.Points,

Coefficient);

sleep(DELAY);

if i <> PruferTree.EdgesCreatingOrder.Count - 1 then

AddUsed(Texts, PruferTree.UsedSons[i] + 1);

sleep(DELAY);

end;

GraphVisualComponents.Destroy;

TreeVisualComponents.Destroy;

BUILDING := false;

end);

end;

procedure VTP(PruferVisual: TPruferVisual; Tree: TTree; GT: TCompleteComponents; L: TLabel; t: TText);

var

i, index, ancestor: integer;

gc, tc: TCircle;

gl, tl: TLine;

Lines: TList<TLine>;

Circles: TList<TCircle>;

vp: TPruferTreeVerticesPair;

begin

BUILDING := true;

Lines := TList<TLine>.Create;

Circles := TList<TCircle>.Create;

for i := 0 to Length(PruferVisual.PruferCode.Code) - 1 do

begin

sleep(DELAY);

vp := TPruferTreeVerticesPair.Create(Tree.Vertices[PruferVisual.Pairs[i].V1], Tree.Vertices[PruferVisual.Pairs[i].V2]);

index := vp.SonVertex.Id;

ancestor := vp.ParentVertex.Id;

gc := GT.Graph.Circles[PruferVisual.Pairs[i].V2];

gl := GT.Graph.Lines[ancestor.ToString + ',' + index.ToString];

tc := GT.Tree.Circles[Tree.IndexOfVertex(PruferVisual.Pairs[i].V2)];

tl := GT.Tree.Lines[ancestor.ToString + ',' + index.ToString];

Lines.Add(gl);

Lines.Add(tl);

Circles.Add(gc);

Circles.Add(tc);

TThread.Synchronize(nil,

procedure

begin

gc.Fill.Color := TAlphaColorRec.Red;

tc.Fill.Color := TAlphaColorRec.Red;

t.Text := t.Text + PruferVisual.PruferCode.Code[i].ToString;

if i <> Length(PruferVisual.PruferCode.Code) - 1 then

t.Text := t.Text + ', ';

end);

sleep(DELAY);

TThread.Synchronize(nil,

procedure

begin

gc.Fill.Color := TAlphaColorRec.Black;

tc.Fill.Color := TAlphaColorRec.Black;

gc.Opacity := 0.2;

tc.Opacity := 0.2;

gl.Opacity := 0.2;

tl.Opacity := 0.2;

end);

end;

sleep(DELAY);

TThread.Synchronize(nil,

procedure

var

line: TLine;

circle: TCircle;

begin

L.Text := PruferVisual.PruferCode.ToString;

for line in Lines do

line.Opacity := 1;

for circle in Circles do

circle.Opacity := 1;

end);

Lines.Destroy;

Circles.Destroy;

BUILDING := false;

end;

procedure VisualizeToPrufer(PruferVisual: TPruferVisual; Tree: TTree; GT: TCompleteComponents; L: TLabel);

var

t: TText;

i: TImage;

TreeCoefficient: TTreeCoefficient;

begin

t := TText.Create(GT.Tree.Circles.First.Owner);

i := GT.Tree.Circles.First.Parent as TImage;

TreeCoefficient := GetCoefficientForTree(i, Tree);

with t do

begin

Parent := i;

Width := i.Width \* 0.8;

Height := TreeCoefficient.Other / 3;

Font.Size := TreeCoefficient.Other / 4;

VertTextAlign := TTextAlign.Center;

HorzTextAlign := TTextAlign.Leading;

Position.Point := TPointF.Create(i.Width \* 0.1, i.Height - 1.5 \* Height);

Text := 'Код: ';

end;

TTask.Run(

procedure

begin

VTP(PruferVisual, Tree, GT, L, t);

end);

end;

{ TTexts }

constructor TTexts.Create(Image: TImage; Coefficient: TTreeCoefficient);

begin

Self.Coefficient := Coefficient;

MinText := TText.Create(Image);

with MinText do

begin

Parent := Image;

Width := Image.Width \* 0.8;

Height := Coefficient.Other / 3;

Font.Size := Coefficient.Other / 4;

VertTextAlign := TTextAlign.Center;

HorzTextAlign := TTextAlign.Leading;

Position.Point := TPointF.Create(Image.Width \* 0.1, Image.Height - 2.5 \* MinText.Height);

Text := 'Минимум - ';

end;

UsedText := TText.Create(Image);

with UsedText do

begin

Parent := Image;

Width := Image.Width \* 0.8;

Height := Coefficient.Other / 3;

Font.Size := Coefficient.Other / 4;

VertTextAlign := TTextAlign.Center;

HorzTextAlign := TTextAlign.Leading;

Position.Point := TPointF.Create(Image.Width \* 0.1, Image.Height - 1.5 \* MinText.Height);

Text := 'Использованные: ';

end;

end;

procedure TTexts.Destroy;

begin

MinText.Destroy;

UsedText.Destroy;

end;

end.