**Реферат**

Представляемый документ содержит:

52 страницы текста и список из двух источников.

Объектом исследования является структура данных «Q-дерево».

Цель работы состоит в создании программного комплекса, обеспечивающего работу со структурой данных «Q-дерево», представленной в виде модели. Методы, используемые при разработке, – язык программирования высокого уровня Object Pascal. Созданный программный продукт обеспечивает выполнение всех требований технического задания.

**Содержание**

Введение

1. Техническое задание

1.1 Основание для разработки

1.2 Назначение разработки

1.3 Функциональные требования к программе

1.4 Требования к составу и параметрам технических средств

1.5 Требования к информационной и программной совместимости

1.6 Требования к программной документации

1.7 Порядок контроля и приемки

2. Рабочий проект

2.1 Модуль UnitModel

2.1.1 Назначение

2.1.2 Функциональные требования, реализуемые модулем

2.1.3 Глобальные переменные и константы модуля

2.1.4 Подпрограммы модуля

2.2 Модуль UnitMainForm

2.2.1 Назначение

2.2.2 Функциональные требования, реализуемые модулем

2.2.3 Используемые компоненты

2.2.4 Глобальные переменные и константы модуля

2.2.5 Подпрограммы модуля

Заключение

Список используемых источников

Приложение

# Введение

Цель данной курсовой работы – разработка программного продукта, предназначенного для работы со структурой данных «Q-дерево». Существует множество различных структур данных, предназначенных для работы с множествами: деревья, массивы и так далее. Среди них есть Q-деревья, позволяющие хранить множества точек и обеспечивать к ним быстрый и удобный доступ. Практическое значение. Программный продукт позволяет пользоваться Q-деревьями. Актуальность разработки программного продукта состоит в увеличении скорости работы с множествами. Программный продукт должен быть разработан на языке программирования высокого уровня Object Pascal, использовать принципы объектно-ориентированного программирования и структурный подход к решению поставленных задач.

Результатом выполнения курсовой работы должен стать готовый программный продукт, отвечающий всем требованиям технического задания.

# 1. Техническое задание

## 1.1 Основание для разработки

Основанием для разработки программного продукта служит задание на курсовую работу “Q-дерево точек”.

## 1.2 Назначение разработки

Программный продукт разрабатывается для работы с Q-деревьями точек.

## 1.3 Функциональные требования к программе

1. Возможность добавления элементов в дерево
2. Удаление элементов из дерева
3. Очистка дерева
4. Подсчет количества элементов
5. Отображение элементов дерева в виде точек на карте
6. Поиск точек в заданной прямоугольной области карты
7. Возможность выбора области карты для просмотра содержащихся в ней точек
8. Отображение точек заданной области карты в отдельном окне просмотра
9. Отображение координат выбранных точек

## 1.4 Требования к составу и параметрам технических средств

Программный комплекс должен корректно работать на компьютере со следующими техническими характеристиками:

* процессор Intel® Celeron® CPU 2.40 ГГц;
* оперативная память объемом 512 Мб;
* жесткий диск Seagate ST380011A, объемом 80 Гб;
* видеоадаптер AGP 8X;
* клавиатура;
* манипулятор типа “мышь”.

## 1.5 Требования к информационной и программной совместимости

Для работы программы необходима операционная система Microsoft Windows XP Professional 2002 (SP1-2).

## 1.6 Требования к программной документации

Программная документация должна включать следующие документы:

* техническое задание;
* рабочий проект.

В приложении к документу "Рабочий проект" должен быть приведен листинг исходных текстов программного изделия.

## 1.7 Порядок контроля и приемки

### 1.7.1 Возможность добавления элементов в дерево, подсчет количества элементов

Добавление элементов в дерево производится щелчком левой кнопкой мыши по точке с нужными координатами в окне просмотра (рис. 1)

Рис. 1

Результат: добавление точки в дерево и его перерисовка; увеличение количества точек в дереве на единицу.

### 1.7.2 Удаление элементов из дерева, подсчет количества элементов

Удаление элемента производится путем выделения точки с помощью мыши в окне просмотра в режиме выделения точек и щелчка по кнопке «Удалить точку» (рис. 2)

Рис. 2

Результат: удаление точки из дерева и его перерисовка; уменьшение количества точек в дереве на единицу.

### 1.7.3 Очистка дерева

Очистка дерева (удаление всех элементов) производится щелчком по кнопке «Удалить все» (рис. 3)

Рис. 3

Результат: удаление всех элементов дерева и соответствующая перерисовка изображений

### 1.7.4 Возможность выбора прямоугольной области карты для просмотра содержащихся в ней точек, поиск точек в заданной прямоугольной области карты

Выбор области просмотра осуществляется перемещением окна выделения с помощью мыши или клавиш (рис. 4)

Рис. 4

Результат: перемещение окна выделения, поиск и отрисовка точек, находящихся в выделенной области карты.

###

### 1.7.5 Отображение элементов дерева в виде точек на карте, отображение координат выбираемых точек

Выбор точки производится с помощью щелчка левой кнопкой мыши по точке с нужными координатами в режиме выбора точек (рис. 5)

Рис. 5

Результат: отображение координат выбранной точки в строке состояния; перерисовка соответствующим цветом ее изображения в окне просмотра.

### 1.7.6 Отображение точек заданной области карты в отдельном окне просмотра, отображение координат выбираемых точек

Для получения координат точки без ее выделения достаточно навести указатель мыши на ее изображение в окне просмотра (рис. 6)

Рис. 6

Результат: отображение координат точки в строке состояния без ее выбора; перерисовка соответствующим цветом ее изображения в окне просмотра.

# 2. Рабочий проект

## 2.1 Модуль UnitModel

### 2.1.1 Назначение

Данный модуль представляет собой реализацию модели структуры данных «Q-дерево точек».

### 2.1.2 Функциональные требования, реализуемые модулем

* Возможность добавления элементов в дерево
* Удаление элементов из дерева
* Очистка дерева
* Поиск точек в заданной прямоугольной области карты.

### 2.1.3 Глобальные переменные и константы модуля

Константы

* М = 3 – максимальное число точек в листе;
	+ - тип – целый;
		- область видимости – внутри и вне модуля;
		- используется в операциях вставки и удаления элементов дерева для проверки числа точек в листьях.

### 2.1.4 Подпрограммы модуля

#### **2.1.4.1 Функция InsertPoint**

* Функция предназначена для вставки нового элемента в Q-дерево
* Параметры
* выходной параметр – указатель на узел дерева, в которое вставляется элемент (тип PNode);
* входной параметр – границы этого узла (тип TRect);
* входной параметр – координаты вставляемой точки (тип TPoint);
* Функция возвращает логическое значение (тип boolean), указывающее на изменение количества элементов в дереве
* Локальные переменные
* CurNode – текущий квадрант (тип PNode);
* DopArray – дополнительный массив, необходимый при делении листа на новые узлы (тип TArrayOfPoints);
* midX, midY – координаты середины узла (тип real);
* NewBounds – границы нового узла, передаваемые в качестве параметра в рекурсивном вызове функции (тип TRect);
* i – счетчик цикла (тип integer).
* Словесный алгоритм

В начале своей работы функция проверяет, не является ли пустым параметр-указатель; если да, то для него выделяется память, устанавливаются начальные значения полей и вставляется новый элемент. Если он не является листом, осуществляется цикл переходов к листу с нужными границами. Далее проверяется число элементов в листе, и, если оно меньше допустимого, туда вставляется новая точка; иначе этот лист разделяется на 4 новых, его точки рекурсивно распределяются по новым листам и затем – вставка новой точки.

####

#### **2.1.4.2 Процедура DeletePoint**

* Процедура предназначена для удаления элемента из Q-дерева
* Параметры
* выходной параметр – указатель на корневой узел дерева, из которого удаляется элемент (тип PNode);
* входной параметр – границы этого узла (тип TRect);
* входной параметр – координаты вставляемой точки (тип TPoint);
* Предусловия

Указатель на дерево не должен быть пустым

* Локальные переменные
* CurNode – текущий квадрант (тип PNode);
* ParentNode – родительский узел листа с удаляемой точкой;
* DopArray – дополнительный массив, необходимый при делении листа на новые узлы (тип TArrayOfPoints);
* midX, midY – координаты середины узла (тип real);
* PointsInNodes, numSZ, numSV, numYZ, numYV – переменные, использующиеся при подсчете числа точек в листах (тип real);
* there – индикатор наличия точки в дереве (тип boolean);
* N – число точек в листе (тип integer);
* i – счетчик цикла (тип integer).
* Словесный алгоритм

В начале своей работы функция проверяет, не является ли пустым параметр-указатель; если да – выход из подпрограммы. Если он не является листом, осуществляется цикл переходов к листу с нужными границами. Далее проверяется наличие точки в листе, и, если она там не обнаружена, процедура заканчивает свою работу; иначе происходит удаление точки из листа и последующая проверка общего числа точек в соседних листах. Если появилась возможность, соседние листы объединяются в один, старые удаляются.

#### **2.1.4.3 Процедура ClearTree**

* Процедура предназначена для удаления всех элементов Q-дерева
* Параметры
* выходной параметр – указатель на узел дерева (тип PNode);
* Предусловия

Указатель на дерево не должен быть пустым

* Словесный алгоритм

В начале своей работы функция проверяет, не является ли пустым параметр-указатель; если да – выход из подпрограммы. Если он не является листом, осуществляются рекурсивные вызовы подпрограммы для каждого из его дочерних узлов; если параметр-указатель является листом, подпрограмма освобождает занятую им память и завершает свою работу.

#### **2.1.4.4 Функция Find**

* Функция предназначена для поиска элементов Q-дерева, расположенных в заданной области карты
* Параметры
* входной параметр – указатель на узел дерева (тип PNode);
* параметр-константа – границы этого узла (тип TRect);
* параметр-константа – границы заданной области карты (тип TRect);
* Функция возвращает список (тип TList) элементов дерева, расположенных в заданной области
* Предусловия

Указатель на дерево не должен быть пустым

* Локальные переменные
* NewBounds – границы нового узла, передаваемые в качестве параметра в рекурсивном вызове функции (тип TRect);
* i – счетчик цикла (тип integer).
* Словесный алгоритм

В начале своей работы функция проверяет, не является ли пустым параметр-указатель; если да – выход из подпрограммы. Если часть площади узла находится в заданной области, осуществляются рекурсивные вызовы подпрограммы для каждого из его дочерних узлов. Для достигнутых таким образом листьев происходит проверка точек на принадлежность заданной области.

#### **2.1.4.5 Процедура SetProperties**

* Процедура предназначена для выделения памяти и установки начальных характеристик для нового узла
* Параметры
* выходной параметр – указатель на узел дерева (тип PNode);
* Словесный алгоритм

Для нового узла, переданного в качестве параметра, выделяется память, устанавливаются начальные характеристики: тип узла (лист) и количество точек в нем (0).

* Подпрограмма используется функцией вставки точек в дерево при разделении листа на 4 новых.

#### **2.1.4.6 Процедура CopyPoints**

* Процедура предназначена для копирования точек из листа в дополнительный массив
* Параметры
* входной параметр – указатель на узел дерева, из которого происходит копирование (тип PNode);
* выходной параметр – дополнительный массив, необходимый при делении листа на новые узлы (тип TArrayOfPoints);
* выходной параметр – счетчик элементов в дополнительном массиве (тип integer).
* Локальные переменные
* j – счетчик цикла (тип integer).
* Словесный алгоритм

Подпрограмма копирует значения точек из данного листа в дополнительный массив, одновременно увеличивая число его элементов, передаваемое в качестве параметра.

* Подпрограмма используется функцией удаления точек из дерева при объединении 4-х листов в один.

## 2.2 Модуль UnitMainForm

### 2.2.1 Назначение

В данном модуле описаны методы работы с Q-деревом точек

### 2.2.2 Функциональные требования, реализуемые модулем

* Подсчет количества элементов в дереве
* Отображение элементов дерева в виде точек на карте
* Возможность выбора области карты для просмотра содержащихся в ней точек
* Отображение точек заданной области карты в отдельном окне просмотра
* Отображение координат выбранных точек

### 2.2.3 Используемые компоненты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Имякомпонента | Класс | Настраиваемыесвойства | Значения | Обработанные события |
| 1 | MainForm | TMainForm | BorderStyle | bsSingle | OnCreate;OnKeyDown |
| Caption | Q-дерево  |
| KeyPreview | True |
| 2 | MaxImage | TImage | – | – | OnCreate;OnMouseMove |
| 3 | MinImage | TImage | – | – | – |
| 4 | ShapeView | TShape | Brush | Style | bsClear | OnMouseDown;OnMouseMove;OnMouseUp |
| Pen | Color | clRed |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Имякомпонента | Класс | Настраиваемыесвойства | Значения | Обработанные события |
| 5 | SBtnCursor | TSpeedButton | Down | True | – |
| GroupIndex | 1 |
| 6 | SBtnPoints | TSpeedButton | GroupIndex | 1 | – |
| 7 | ButtonDelete | TBitBtn | Caption | Удалить точку | OnClick |
| Enabled | False |
| ShowHint | True |
| Hint | Удалить выбранную точку |
| 8 | ButtonClear | TBitBtn | Caption | Удалить все | OnClick |
| ShowHint | True |
| Hint | Удалить все точки дерева |
| 9 | StatusBar | TStatusBar | – | – | – |

### 2.2.4 Глобальные переменные и константы модуля

Константы

* Xmax = 1024 – ширина всего квадрата, отведенного под Q-дерево;
	+ тип – целый;
	+ область видимости – внутри и вне модуля;
	+ используется в операциях вставки и удаления элементов для задания границ главного квадранта
		- K = 10.56 – отношение длины стороны окна выделения к длине стороны окна просмотра;
	+ тип – вещественный;
	+ область видимости – внутри модуля;
	+ используется при выводе на карту изображений точек
		- R = 3 – радиус точки, изображенной на карте;
	+ тип – целый;
	+ область видимости – внутри модуля;
	+ используется при выводе изображений точек
		- LightColor = clYellow – цвет подсветки точек;
	+ тип – TColor;
	+ область видимости – внутри модуля;
	+ используется при выводе изображений точек
		- SelectColor = clRed – цвет выделенной точки;
	+ тип – TColor;
	+ область видимости – внутри модуля;
	+ используется при выводе изображений точек
		- BackColor = clBtnFace – цвет фона карты;
	+ тип – TColor;
	+ область видимости – внутри модуля;
	+ используется при выводе изображений точек.

Переменные

* + - Tree – указатель на корневой узел дерева;
	+ тип – PNode;
	+ область видимости – внутри модуля;
	+ используется в подпрограммах, работающих с деревом.
		- X0, Y0 – начальные координаты указателя мыши при перемещении окна выделения;
	+ тип – целый;
	+ область видимости – внутри модуля;
	+ используются при определении координат просматриваемой области карты
		- drag = false – индикатор перетаскивания окна выделения;
	+ тип – логический;
	+ область видимости – внутри модуля;
	+ используется при определении координат просматриваемой области карты
		- PointCount = 0 – количество точек в дереве;
	+ тип – целый;
	+ область видимости – внутри модуля;
	+ используется для определения числа точек в дереве
		- mainBounds, Query – координаты соответственно главного квадранта и выделенной области;
	+ тип – TRect;
	+ область видимости – внутри модуля;
	+ используются при поиске и выводе изображений точек просматриваемой области
		- LightPoint, SelectedPoint – соответственно текущая и выделенная точки;
	+ тип – TPoint;
	+ область видимости – внутри модуля;
	+ используются для выбора и удаления точек.

### 2.2.5 Подпрограммы модуля

#### **2.2.5.1 Процедура DrawPoint**

* + Процедура предназначена для вывода изображений точек на карту
	+ Процедура является методом класса TMainForm
	+ Параметры
* параметр-константа – точка (тип TPoint);
* входной параметр – цвет изображенной точки (тип TColor);
	+ Локальные переменные
* dopX, dopY – координаты точки относительно окна просмотра (тип integer).
	+ Словесный алгоритм

Процедура вычисляет координаты отображаемой точки для каждой из карт (большой и малой) и рисует точку в виде эллипса радиусом R.

#### **2.2.5.2 Процедура ClearBackground**

* + Процедура стирает предыдущее изображение на карте
	+ Процедура является методом класса TMainForm
	+ Параметры
* входной параметр – компонент-карта (тип TImage);
	+ Словесный алгоритм

Процедура закрашивает поверхность карты цветом фона BackColor.

#### **2.2.5.3 Процедура DrawRegion**

* + Процедура предназначена для поиска и вывода изображений точек дерева в заданной области карты
	+ Процедура является методом класса TMainForm
	+ Параметры
* параметр-константа – указатель на узел дерева (тип PNode);
* параметр-константа – границы заданной области (тип TRect);
	+ Локальные переменные
* FindedPoints – список найденных точек (тип TList);
* dopPoint – точка из списка (тип TPoint);
* i – счетчик цикла (тип integer).
	+ Словесный алгоритм

Процедура создает пустой список, копирует туда точки дерева, найденные в заданной области, и выводит их изображения на карты.

#### **2.2.5.4 Процедура FormCreate**

* + Процедура предназначена для задания начальных координат областей и точек
	+ Процедура является методом класса TMainForm
	+ Параметры
* входной параметр – объект, сгенерировавший событие (тип TObject)
	+ Словесный алгоритм

Процедура устанавливает границы главного квадранта и выделенной области, начальные координаты для текущей и выбранной точек.

#### **2.2.5.5 Процедура ShapeViewMouseDown**

* + Процедура предназначена для получения начальных координат указателя мыши перед началом перетаскивания выделяющего окна
	+ Процедура является методом класса TMainForm
	+ Параметры
* входной параметр – объект, сгенерировавший событие (тип TObject);
* входной параметр – индикатор нажатой кнопки мыши (тип TMouseButton);
* входной параметр – индикатор нажатой клавиши (тип TShiftState);
* входные параметры – координаты указателя мыши (тип integer)
	+ Словесный алгоритм

Координаты указателя записываются в глобальные переменные X0 и Y0. Индикатору перетаскивания drag присваивается true.

####

#### **2.1.5.6 Процедура ShapeViewMouseUp**

* + Процедура предназначена для установки значения соответствующего индикатора при окончании перетаскивания окна выделения
	+ Процедура является методом класса TMainForm
	+ Параметры
* входной параметр – объект, сгенерировавший событие (тип TObject);
* входной параметр – индикатор нажатой кнопки мыши (тип TMouseButton);
* входной параметр – индикатор нажатой клавиши (тип TShiftState);
* входные параметры – координаты указателя мыши (тип integer)
	+ Словесный алгоритм

Индикатору перетаскивания drag присваивается false.

#### **2.1.5.7 Процедура ShapeViewMouseMove**

* + Процедура предназначена для перемещения окна выделения по малой карте и вывода на карту изображений точек из выделенной области
	+ Процедура является методом класса TMainForm
	+ Параметры
* входной параметр – объект, сгенерировавший событие (тип TObject);
* входной параметр – индикатор нажатой клавиши (тип TShiftState)
* входные параметры – координаты указателя мыши (тип integer)
	+ Предусловия

Индикатор перетаскивания должен быть равен true.

* + Локальные переменные
* newLeft, newTop – новые координаты окна выделения (тип integer)
	+ Словесный алгоритм

Процедура вычисляет новые координаты окна выделения и области просмотра с использованием глобальных переменных X0 и Y0; затем осуществляет поиск и вывод на карту изображений точек из новой области с помощью процедуры DrawRegion.

#### **2.1.5.8 Процедура MaxImageMouseMove**

* + Процедура предназначена для отображения координат выделяемых точек в строке состояния и выделения их изображений на карте
	+ Процедура является методом класса TMainForm
	+ Параметры
* входной параметр – объект, сгенерировавший событие (тип TObject);
* входной параметр – индикатор нажатой клавиши (тип TShiftState);
* входные параметры – координаты указателя мыши (тип integer)
	+ Локальные переменные
* Point – выделенная точка (тип TPoint);
* Rect – область поиска точки в дереве (тип TRect);
* str – строка с координатами выбранной точки (тип string);
* List – список точек, найденных в области вблизи указателя мыши
	+ Словесный алгоритм

Подпрограмма выводит в строку состояния координаты движущегося указателя мыши и осуществляет проверку того, наведен ли он на точку, путем поиска точек дерева в области вокруг указателя. Если таковые имеются, изображение первой из них перерисовывается соответствующим цветом.

#### **2.1.5.9 Процедура MaxImageClick**

* + Процедура предназначена для добавления точки в дерево и «запоминания» координат выбранной точки
	+ Процедура является методом класса TMainForm
	+ Параметры
* входной параметр – объект, сгенерировавший событие (тип TObject)
	+ Локальные переменные
* Point – новая либо выбранная точка (тип TPoint);
* str – строка с координатами выбранной точки (тип string);
* i, j – координаты точки относительно окна просмотра (тип integer)
	+ Словесный алгоритм

Подпрограмма получает координаты новой (или выбранной) точки из строки состояния. Затем, если программа находится в режиме добавления точек, вставляет в дерево новую точку; в зависимости от результата функции вставки, увеличивает счетчик точек на единицу и перерисовывает изображение. В режиме выбора точек процедура записывает в глобальную переменную координаты выбранной точки и перекрашивает ее на карте соответствующим цветом. Координаты выбранной точки выводятся в строку состояния.

#### **2.1.5.10 Процедура ButtonDeleteClick**

* + Процедура предназначена для удаления выбранной точки из дерева
	+ Процедура является методом класса TMainForm
	+ Параметры
* входной параметр – объект, сгенерировавший событие (тип TObject)
	+ Словесный алгоритм

Подпрограмма удаляет выбранную точку из дерева; затем, если необходимо, перерисовывает просматриваемую область карты.

#### 2.1.5.11 Процедура ButtonClearClick

* + Процедура предназначена для удаления всех точек из дерева
	+ Процедура является методом класса TMainForm
	+ Параметры
* входной параметр – объект, сгенерировавший событие (тип TObject)
	+ Словесный алгоритм

Подпрограмма удаляет все точки из дерева, «стирает» изображение с карты и устанавливает «пустые » координаты для выбранной и текущей точек.

#### **2.1.5.12 Процедура FormKeyDown**

* + Процедура осуществляет перемещение окна выделения при нажатии клавиш
	+ Процедура является методом класса TMainForm
	+ Параметры
* входной параметр – объект, сгенерировавший событие (тип TObject);
* выходной параметр – индикатор нажатой клавиши (тип word);
* входной параметр – индикатор нажатой клавиши (тип TShiftState)
	+ Локальные константы

– dif = 4 – число пикселей, на которое перемещается окно выделения

* + Словесный алгоритм

Подпрограмма вызывает перемещающую окно выделения процедуру ShapeViewMouseMove, передавая ей разные параметры в зависимости от нажатой клавиши.

# Заключение

Разработанный программный продукт обеспечивает выполнение всех требований, предъявленных к нему в техническом задании.

Программный продукт рекомендован к использованию для широкого круга пользователей. Использование программного продукта позволяет существенно облегчить работу с множествами и ускорить их обработку.

# Список используемых источников

1. Сухарев М.В. Основы Delphi. Профессиональный подход – СПб.: Наука и Техника, 2004.
2. Кэнту М. Delphi 7: для профессионалов – СПб.: Питер, 2004.

# Приложение

Текст программы

program Qtree;

uses Forms,

 UnitMainForm in 'UnitMainForm.pas' {MainForm},

 UnitModel in 'UnitModel.pas';

{$R \*.res}

begin

 Application.Initialize;

 Application.CreateForm(TMainForm, MainForm);

 Application.Run;

end.

unit UnitModel;

interface

uses Classes;

const M = 3; //число точек в листе

type

 //Тип узла дерева-----------------------------------

 TNodeKind = (nkBranch, nkLeaf);

 TPoint = record

 X: real;

 Y: real;

 end;

 TRect = record

 X1, Y1, X2, Y2: real;

 end;

 //Массив для хранения точек в листе-----------------

 TArrayOfPoints = array[1..M] of TPoint;

 //Узел дерева---------------------------------------

 PNode = ^TNode;

 TNode = packed record

 case Kind: TNodeKind of

 nkBranch: (SZ, SV, YZ, YV: PNode);

 nkLeaf: (Points: TArrayOfPoints;

 PointsCount: integer);

 end;

function InsertPoint(var Node: PNode; Bounds: TRect; Point: TPoint): boolean;

procedure DeletePoint(var Node: PNode; Bounds: TRect; Point: TPoint);

procedure ClearTree(var Node: PNode);

function Find(Node: PNode; const Bounds, Rect: TRect): TList;

implementation

//Установка характеристик нового листа =======================================

procedure SetProperties(var ChildNode: PNode);

begin

New(ChildNode);

ChildNode^.Kind:= nkLeaf;

ChildNode^.PointsCount:= 0; //в массиве нет точек

end;

//Копирование точек из листа в дополнительный массив =========================

procedure CopyPoints(Node: PNode; var DopArray: TArrayOfPoints; var i: integer);

var j: integer;

begin

for j:=1 to Node^.PointsCount do

 begin

 DopArray[i]:= Node^.Points[j];

 inc(i);

 end;

end;

//ВСТАВКА ТОЧКИ В ДЕРЕВО =====================================================

function InsertPoint(var Node: PNode; Bounds: TRect; Point: TPoint): boolean;

var CurNode: PNode; //текущий квадрант

 DopArray: TArrayOfPoints; //дополнительный массив (когда делим узел)

 i: integer;

 midX, midY: real;

 NewBounds: TRect;

begin

if Node = nil then

 begin

 New(Node);

 Node^.Kind:= nkLeaf;

 Node^.PointsCount:= 0;

 end;

CurNode:= Node;

Result:= true;

with Bounds do

 begin

 while CurNode^.Kind = nkBranch do //если ветвь, то смотрим, куда идти

 begin

 midX:= (X2 - X1)/2 + X1;

 midY:= (Y2 - Y1)/2 + Y1;

 if Point.X < midX then

 if Point.Y < midY then

 begin

 CurNode:= CurNode^.SZ;

 X2:= midX;

 Y2:= midY;

 end

 else

 begin

 CurNode:= CurNode^.YZ;

 Y1:= midY;

 X2:= midX;

 end

 else

 if Point.Y < midY then

 begin

 CurNode:= CurNode^.SV;

 X1:= midX;

 Y2:= midY;

 end

 else

 begin

 CurNode:= CurNode^.YV;

 X1:= midX;

 Y1:= midY;

 end;

 end;

 midX:= (X2 - X1)/2 + X1;

 midY:= (Y2 - Y1)/2 + Y1;

 end;

//Собственно вставка----------------------------------------------------------

//Проверить, есть ли место в массиве точек и нет ли уже там новой:

for i:=1 to CurNode^.PointsCount do

 if (CurNode^.Points[i].X = Point.X)and(CurNode^.Points[i].Y = Point.Y) then

 begin

 Result:= false;

 Exit;

 end;

//Если массив не заполнен, вставляем точку...

if CurNode^.PointsCount < M then

 begin

 CurNode^.Points[CurNode^.PointsCount + 1]:= Point;

 CurNode^.PointsCount:= CurNode^.PointsCount + 1;

 end

else

 begin

 //...иначе делим лист на 4 новых:

 DopArray:= CurNode^.Points;

 CurNode^.Kind:= nkBranch;

 SetProperties(CurNode^.SZ);

 SetProperties(CurNode^.SV);

 SetProperties(CurNode^.YZ);

 SetProperties(CurNode^.YV);

 //Распределение точек по узлам

 for i:=1 to M do

 with Bounds do

 if DopArray[i].X < midX then

 if DopArray[i].Y < midY then

 begin

 NewBounds.X1:= X1;

 NewBounds.X2:= (X2 - X1)/2 + X1;

 NewBounds.Y1:= Y1;

 NewBounds.Y2:= (Y2 - Y1)/2 + Y1;

 InsertPoint(CurNode^.SZ, NewBounds, DopArray[i]);

 end

 else

 begin

 NewBounds.X1:= X1;

 NewBounds.X2:= (X2 - X1)/2 + X1;

 NewBounds.Y1:= (Y2 - Y1)/2 + Y1;

 NewBounds.Y2:= Y2;

 InsertPoint(CurNode^.YZ, NewBounds, DopArray[i]);

 end

 else

 if DopArray[i].Y < midY then

 begin

 NewBounds.X1:= (X2 - X1)/2 + X1;

 NewBounds.X2:= X2;

 NewBounds.Y1:= Y1;

 NewBounds.Y2:= (Y2 - Y1)/2 + Y1;

 InsertPoint(CurNode^.SV, NewBounds, DopArray[i]);

 end

 else

 begin

 NewBounds.X1:= (X2 - X1)/2 + X1;

 NewBounds.X2:= X2;

 NewBounds.Y1:= (Y2 - Y1)/2 + Y1;

 NewBounds.Y2:= Y2;

 InsertPoint(CurNode^.YV, NewBounds, DopArray[i]);

 end;

 //Вставка новой точки

 InsertPoint(CurNode, Bounds, Point);

 end;

end;

//УДАЛЕНИЕ ТОЧКИ ИЗ ДЕРЕВА ===================================================

procedure DeletePoint(var Node: PNode; Bounds: TRect; Point: TPoint);

var CurNode, ParentNode: PNode;

 DopArray: TArrayOfPoints;

 midX, midY, PointsInNodes, numSZ, numSV, numYZ, numYV: real;

 there: boolean;

 i, N: integer;

begin

if Node = nil then

 Exit;

CurNode:= Node;

ParentNode:= CurNode;

with Bounds do

 while CurNode^.Kind = nkBranch do //если ветвь, то смотрим, куда идти

 begin

 ParentNode:= CurNode;

 midX:= (X2 - X1)/2 + X1;

 midY:= (Y2 - Y1)/2 + Y1;

 if Point.X < midX then

 if Point.Y < midY then

 begin

 CurNode:= CurNode^.SZ;

 X2:= midX;

 Y2:= midY;

 end

 else

 begin

 CurNode:= CurNode^.YZ;

 Y1:= midY;

 X2:= midX;

 end

 else

 if Point.Y < midY then

 begin

 CurNode:= CurNode^.SV;

 X1:= midX;

 Y2:= midY;

 end

 else

 begin

 CurNode:= CurNode^.YV;

 X1:= midX;

 Y1:= midY;

 end;

 end;

//Собственно удаление-------------------------------------------------------

N:= CurNode^.PointsCount;

//Проверить, есть ли в массиве удаляемая точка:

there:= false;

for i:=1 to M do

 if (CurNode^.Points[i].X = Point.X)and(CurNode^.Points[i].Y = Point.Y) then

 begin

 there:= true;

 break;

 end;

//Удаляем точку (либо выходим, если таковой не имеется)

if there then

 begin

 CurNode^.Points[i]:= CurNode^.Points[N];

 CurNode^.PointsCount:= CurNode^.PointsCount - 1;

 end

else Exit;

if Node^.Kind = nkLeaf then

 Exit;

//Посмотрим, можно ли объединить соседние узлы

numSZ:= ParentNode^.SZ^.PointsCount;

numSV:= ParentNode^.SV^.PointsCount;

numYZ:= ParentNode^.YZ^.PointsCount;

numYV:= ParentNode^.YV^.PointsCount;

PointsInNodes:= numSZ + numSV + numYZ + numYV;

if PointsInNodes <= M then

 begin

 //Точки из всех листьев переносим в вышестоящий узел

 i:=1;

 CopyPoints(ParentNode^.SZ, DopArray, i);

 CopyPoints(ParentNode^.SV, DopArray, i);

 CopyPoints(ParentNode^.YZ, DopArray, i);

 CopyPoints(ParentNode^.YV, DopArray, i);

 //Удаляем старые листья

 Dispose(ParentNode^.SZ);

 Dispose(ParentNode^.SV);

 Dispose(ParentNode^.YZ);

 Dispose(ParentNode^.YV);

 ParentNode^.Kind:= nkLeaf;

 ParentNode^.Points:= DopArray;

 end;

end;

//УДАЛЕНИЕ ДЕРЕВА ============================================================

procedure ClearTree(var Node: PNode);

begin

if Node = nil then

 Exit;

if Node^.Kind = nkBranch then

 begin

 ClearTree(Node^.SZ);

 ClearTree(Node^.SV);

 ClearTree(Node^.YZ);

 ClearTree(Node^.YV);

 end;

Dispose(Node);

Node:= nil;

end;

//ПОИСК ТОЧЕК В ЗАДАННОЙ ОБЛАСТИ =============================================

function Find(Node: PNode; const Bounds, Rect: TRect): TList;

var NewBounds: TRect;

 i: integer;

begin

Result:= TList.Create;

if Node = nil then

 Exit;

with Bounds do

 if (X2 >= Rect.X1)and(X1 <= Rect.X2)and(Y2 >= Rect.Y1)and(Y1 <= Rect.Y2) then

 if Node^.Kind = nkBranch then

 begin

 NewBounds.X1:= X1;

 NewBounds.X2:= (X2 - X1)/2 + X1;

 NewBounds.Y1:= Y1;

 NewBounds.Y2:= (Y2 - Y1)/2 + Y1;

 Result.Assign(Find(Node^.SZ, NewBounds, Rect), laOr);

 NewBounds.X1:= (X2 - X1)/2 + X1;

 NewBounds.X2:= X2;

 NewBounds.Y1:= Y1;

 NewBounds.Y2:= (Y2 - Y1)/2 + Y1;

 Result.Assign(Find(Node^.SV, NewBounds, Rect), laOr);

 NewBounds.X1:= X1;

 NewBounds.X2:= (X2 - X1)/2 + X1;

 NewBounds.Y1:= (Y2 - Y1)/2 + Y1;

 NewBounds.Y2:= Y2;

 Result.Assign(Find(Node^.YZ, NewBounds, Rect), laOr);

 NewBounds.X1:= (X2 - X1)/2 + X1;

 NewBounds.X2:= X2;

 NewBounds.Y1:= (Y2 - Y1)/2 + Y1;

 NewBounds.Y2:= Y2;

 Result.Assign(Find(Node^.YV, NewBounds, Rect), laOr);

 end

 else

 begin

 for i:=1 to Node^.PointsCount do

 if (Node^.Points[i].X >= Rect.X1)and

 (Node^.Points[i].X <=Rect.X2)and

 (Node^.Points[i].Y >= Rect.Y1)and

 (Node^.Points[i].Y <= Rect.Y2) then

 Result.Add(@(Node^.Points[i]));

 end;

end;

end.

unit UnitMainForm;

interface

uses

 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

 Dialogs, ExtCtrls, StdCtrls, UnitModel, ComCtrls, Buttons;

const Xmax = 1024; //ширина всего квадрата, отведенного под квадродерево

type

 TMainForm = class(TForm)

 MaxImage: TImage;

 ShapeMax: TShape;

 MinImage: TImage;

 ShapeView: TShape;

 Shape3: TShape;

 LabelTop: TLabel;

 LabelLeft: TLabel;

 LabelRight: TLabel;

 LabelBottom: TLabel;

 StatusBar: TStatusBar;

 SBtnCursor: TSpeedButton;

 SBtnPoints: TSpeedButton;

 ButtonClear: TBitBtn;

 ButtonDelete: TBitBtn;

 procedure DrawPoint(const Point: TPoint; PointColor: TColor);

 procedure ClearBackground(Image: TImage);

 procedure DrawRegion(const Node: PNode; const Bounds: TRect);

 procedure ShapeViewMouseDown(Sender: TObject; Button: TMouseButton;

 Shift: TShiftState; X, Y: Integer);

 procedure ShapeViewMouseUp(Sender: TObject; Button: TMouseButton;

 Shift: TShiftState; X, Y: Integer);

 procedure ShapeViewMouseMove(Sender: TObject; Shift: TShiftState; X,

 Y: Integer);

 procedure MaxImageMouseMove(Sender: TObject; Shift: TShiftState; X,

 Y: Integer);

 procedure MaxImageClick(Sender: TObject);

 procedure FormCreate(Sender: TObject);

 procedure ButtonClearClick(Sender: TObject);

 procedure ButtonDeleteClick(Sender: TObject);

 procedure FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);

 private

 { Private declarations }

 public

 { Public declarations }

 end;

var

 MainForm: TMainForm;

implementation

{$R \*.dfm}

const K = 10.56; //масштаб (MaxImage.Width/MinImage.Width)

 R = 3; //радиус точки на форме

 LightColor = clLime; //цвет подсветки точек

 SelectColor = clRed; //цвет выделенной точки

 BackColor = clWhite; //цвет фона

var Tree: PNode;

 X0, Y0: integer;

 drag: boolean = false; //флажок перетаскивания окна просмотра

 PointCount: integer = 0; //число точек в дереве

 mainBounds, Query: TRect; //главный квадрант и окно просмотра

 LightPoint, SelectedPoint: TPoint;

//Отрисовка точки ============================================================

procedure TMainForm.DrawPoint(const Point: TPoint; PointColor: TColor);

var dopX, dopY: integer;

begin

//В большом окне...

with Point do

 begin

 with MaxImage.Canvas do

 begin

 Brush.Color:= PointColor;

 Brush.Style:= bsSolid;

 Pen.Color:= PointColor;

 dopX:= round(X - Query.X1);

 dopY:= round(Y - Query.Y1);

 Ellipse(dopX-R, dopY-R, dopX+R, dopY+R);

 end;

//...и в малом:

 with MinImage.Canvas do

 begin

 Brush.Color:= PointColor;

 Brush.Style:= bsSolid;

 Pen.Color:= PointColor;

 Ellipse(round(X/K)-1, round(Y/K)-1, round(X/K)+1, round(Y/K)+1);

 end;

 end;

end;

//"Очистка" фона =============================================================

procedure TMainForm.ClearBackground(Image: TImage);

begin

with Image.Canvas do

 begin

 Brush.Style:= bsSolid;

 Brush.Color:= BackColor;

 Rectangle(-1,-1,Image.Width + 1,Image.Height + 1);

 end;

end;

//Отрисовка просматриваемой области ==========================================

procedure TMainForm.DrawRegion(const Node: PNode; const Bounds: TRect);

var FindedPoints: TList;

 dopPoint: TPoint;

 i: integer;

begin

FindedPoints:= TList.Create;

with FindedPoints do

 begin

 Assign(Find(Node, mainBounds, Bounds), laOr);

 if Capacity <> 0 then

 for i:= 0 to Count - 1 do

 begin

 dopPoint:= TPoint(FindedPoints[i]^);

 if (dopPoint.X = SelectedPoint.X)and(dopPoint.Y = SelectedPoint.Y) then

 DrawPoint(dopPoint, SelectColor)

 else DrawPoint(dopPoint, clBlack);

 end;

 Free;

 end;

end;

//Задание начальных координат областей и точек ===============================

procedure TMainForm.FormCreate(Sender: TObject);

begin

with mainBounds do

 begin

 X1:= 0;

 Y1:= 0;

 X2:= Xmax;

 Y2:= Xmax;

 end;

with Query do

 begin

 X1:= 0;

 Y1:= 0;

 X2:= MaxImage.Width;

 Y2:= MaxImage.Width;

 end;

with LightPoint do

 begin

 X:= -4;

 Y:= -4;

 end;

with SelectedPoint do

 begin

 X:= -3;

 Y:= -3;

 end;

end;

//НАВИГАЦИЯ В МАЛОМ ОКНЕ =====================================================

procedure TMainForm.ShapeViewMouseDown(Sender: TObject; Button: TMouseButton;

 Shift: TShiftState; X, Y: Integer);

begin

X0:= X;

Y0:= Y;

drag:= true;

end;

procedure TMainForm.ShapeViewMouseUp(Sender: TObject; Button: TMouseButton;

 Shift: TShiftState; X, Y: Integer);

begin

drag:= false;

end;

procedure TMainForm.ShapeViewMouseMove(Sender: TObject; Shift: TShiftState;

 X, Y: Integer);

var newLeft, newTop: integer;

begin

if drag then

 with Sender as TShape do

 begin

 newLeft:= Left + X - X0;

 newTop:= Top + Y - Y0;

 if newLeft + Width >= MinImage.Left + MinImage.Width + 1 then

 newLeft:= MinImage.Left + MinImage.Width + 1 - Width;

 if newLeft <= MinImage.Left - 1 then

 newLeft:= MinImage.Left - 1;

 Left:= newLeft;

 if newTop + Height >= MinImage.Top + MinImage.Height + 1 then

 newTop:= MinImage.Top + MinImage.Height + 1 - Height;

 if newTop <= MinImage.Top - 1 then

 newTop:= MinImage.Top - 1;

 Top:= newTop;

 //Границы просматриваемой области-----------------------------------

 Query.X1:= round((Left - MinImage.Left + 1)\*K);

 Query.X2:= round((Left - MinImage.Left + Width + 1)\*K);

 Query.Y1:= round((Top - MinImage.Top + 1)\*K);

 Query.Y2:= round((Top - MinImage.Top + Height + 1)\*K);

 LabelLeft.Caption:= FloatToStr(Query.X1);

 LabelRight.Caption:= FloatToStr(Query.X2);

 LabelTop.Caption:= FloatToStr(Query.Y1);

 LabelBottom.Caption:= FloatToStr(Query.Y2);

 ClearBackground(MaxImage);

 DrawRegion(Tree, Query);

 end;

end;

//Отображение координат точек в строке состояния =============================

procedure TMainForm.MaxImageMouseMove(Sender: TObject; Shift: TShiftState;

 X, Y: Integer);

var Point: TPoint;

 Rect: TRect;

 str: string[30];

 List: TList;

begin

if SBtnCursor.Down then

 MaxImage.Cursor:= crDefault

else MaxImage.Cursor:= crCross;

with StatusBar do

 with MaxImage.Canvas do

 begin

 //Координаты указателя мыши

 Panels[0].Text := 'X: ' + FloatToStr(X + Query.X1);

 Panels[1].Text := 'Y: ' + FloatToStr(Y + Query.Y1);

 //Если указатель наведен на точку:

 if (Pixels[X,Y] = clBlack)or(Pixels[X,Y] = LightColor)or

 (Pixels[X,Y] = SelectColor) then

 begin

 Point.X:= X + Query.X1;

 Point.Y:= Y + Query.Y1;

 with Point do

 begin

 Rect.X1:= X - R;

 Rect.X2:= X + R;

 Rect.Y1:= Y - R;

 Rect.Y2:= Y + R;

 end;

 List:= TList.Create;

 List.Assign(Find(Tree, mainBounds, Rect), laOr);

 if List.Capacity <> 0 then

 begin

 Point:= TPoint(List[0]^);

 Panels[3].Text:= 'Точка ' + FloatToStr(Point.X) + '; ' +

 FloatToStr(Point.Y);

 //"Подсветка" точки----------------------------------------------

 if Pixels[round(Point.X - Query.X1),round(Point.Y - Query.Y1)] <>

 LightColor then

 with LightPoint do

 begin

 if X >= 0 then

 if (X <> SelectedPoint.X)or(Y <> SelectedPoint.Y) then

 DrawPoint(LightPoint, clBlack)

 else DrawPoint(LightPoint, SelectColor);

 str:= StatusBar.Panels[3].Text;

 X:= StrToFloat(Copy(str, Pos(' ', str)+1, Pos(';', str)-

 Pos(' ', str)-1));

 Y:= StrToFloat(Copy(str, Pos(';', str)+2, 10));

 DrawPoint(LightPoint, LightColor);

 end;

 List.Free;

 end;

 end

 else

 //Долой "подсветку":

 with LightPoint do

 begin

 Panels[3].Text:= '';

 if Tree = nil then

 Exit;

 if Pixels[round(X - Query.X1), round(Y - Query.Y1)] =

 LightColor then

 if (X = SelectedPoint.X)and(Y = SelectedPoint.Y) then

 DrawPoint(LightPoint, SelectColor)

 else DrawPoint(LightPoint, clBlack);

 end;

 end;

end;

//Клик по большому окну ======================================================

procedure TMainForm.MaxImageClick(Sender: TObject);

var Point: TPoint;

 str: string[30];

 i, j: integer;

begin

Point.X:= StrToInt(copy(StatusBar.Panels[0].Text, 4, 10));

Point.Y:= StrToInt(copy(StatusBar.Panels[1].Text, 4, 10));

if SBtnPoints.Down then //В режиме добавления точек -----------------------

 begin

 //Добавление точки в дерево

 if InsertPoint(Tree, mainBounds, Point) then

 PointCount:= PointCount + 1;

 ClearBackground(MaxImage);

 ClearBackground(MinImage);

 //Перерисовка области просмотра

 DrawRegion(Tree, Query);

 DrawRegion(Tree, mainBounds);

 StatusBar.Panels[2].Text:= 'Количество точек: ' + IntToStr(PointCount);

 end

else

 begin

 if (Point.X = SelectedPoint.X)and(Point.Y = SelectedPoint.Y) then

 Exit;

 i:= round(Point.X - Query.X1);

 j:= round(Point.Y - Query.Y1);

 with MaxImage.Canvas do

 begin

 if (Pixels[i,j] = LightColor)or(Pixels[i,j] = clBlack) then

 //"Запомнить" выбранную точку -------------------------------------

 with SelectedPoint do

 begin

 str:= StatusBar.Panels[3].Text;

 if str = '' then

 Exit;

 if X >= 0 then

 DrawPoint(SelectedPoint, clBlack);

 X:= StrToFloat(Copy(str, Pos(' ', str)+1, Pos(';', str)-Pos(' ',

 str)-1));

 Y:= StrToFloat(Copy(str, Pos(';', str)+2, 10));

 StatusBar.Panels[4].Text:= 'Выбрано: ' + FloatToStr(X) + '; ' +

 FloatToStr(Y);

 DrawPoint(SelectedPoint, SelectColor);

 ButtonDelete.Enabled:= true;

 end;

 end;

 end;

end;

//Удаление точки =============================================================

procedure TMainForm.ButtonDeleteClick(Sender: TObject);

begin

DeletePoint(Tree, mainBounds, SelectedPoint);

if (SelectedPoint.X >= Query.X1)and(SelectedPoint.X <= Query.X2)and

 (SelectedPoint.Y >= Query.Y1)and(SelectedPoint.Y <= Query.Y2) then

 begin

 SelectedPoint.X:= -3;

 LightPoint.X:= -4;

 ClearBackground(MaxImage);

 ClearBackground(MinImage);

 DrawRegion(Tree, mainBounds);

 end;

PointCount:= PointCount - 1;

StatusBar.Panels[4].Text:= '';

ButtonDelete.Enabled:= false;

end;

//Удаление дерева ============================================================

procedure TMainForm.ButtonClearClick(Sender: TObject);

begin

ClearTree(Tree);

ClearBackground(MaxImage);

ClearBackground(MinImage);

DrawRegion(Tree, mainBounds);

PointCount:= 0;

with StatusBar do

 begin

 Panels[2].Text:= '';

 Panels[3].Text:= '';

 Panels[4].Text:= '';

 end;

SelectedPoint.X:= -3;

LightPoint.X:= -4;

StatusBar.Panels[4].Text:= '';

ButtonDelete.Enabled:= false;

end;

//Перемещение окошка с помощью клавиш ========================================

procedure TMainForm.FormKeyDown(Sender: TObject; var Key: Word;

 Shift: TShiftState);

const dif = 4;

begin

drag:= true;

with ShapeView do

 begin

 X0:= Left + round(Width/2);

 Y0:= Top + round(Height/2);

 end;

if Key = VK\_UP then

 ShapeViewMouseMove(ShapeView, Shift, X0, Y0 - dif)

else

 if Key = VK\_DOWN then

 ShapeViewMouseMove(ShapeView, Shift, X0, Y0 + dif)

 else

 if Key = VK\_LEFT then

 ShapeViewMouseMove(ShapeView, Shift, X0 - dif, Y0)

 else

 ShapeViewMouseMove(ShapeView, Shift, X0 + dif, Y0);

drag:= false;

end;

end.