# Министерство образования Республики Беларусь

УПРАВЛЕНИЕ ОБРАЗОВАНИЯ МОГИЛЕВСКОГО ОБЛАСТНОГО ИСПОЛНИТЕЛЬНОГО КОМИТЕТА

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«Могилевский государственный политехнический колледж»

# Домашняя контрольная работа №2

# По дисциплине: «Основы алгоритмизации и программирования»

Группа ПО-455

## Выполнил ФИО

Шифр 21

# 2020

**5 Опишите основные компоненты среды Delphi - форма, палитра компонентов, инспектор объектов.**

Ниже перечислены самые основные составные части Delphi:

1. Дизайнер Форм (Form Designer)
2. Окно Редактора Исходного Текста (Editor Window)
3. Палитра Компонент (Component Palette)
4. Инспектор Объектов (Object Inspector)
5. Справочник (On-line help)

Есть, конечно, и другие важные составляющие Delphi, вроде линейки инструментов, системного меню и многие другие, нужные Вам для точной настройки программы и среды программирования.

Программисты на Delphi проводят большинство времени переключаясь между Дизайнером Форм и Окном Редактора Исходного Текста (которое для краткости называют Редактор). Прежде чем Вы начнете, убедитесь, что можете распознать эти два важных элемента. Дизайнер Форм показан на рис.1, окно Редактора - на рис.2.

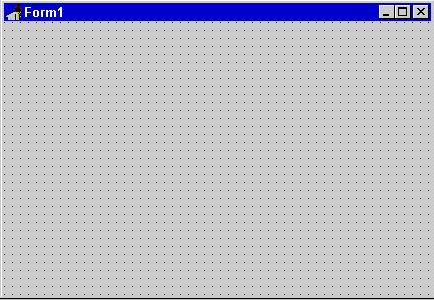


Рисунок 1. Дизайнер Форм - место, где создается визуальный интерфейс программы.

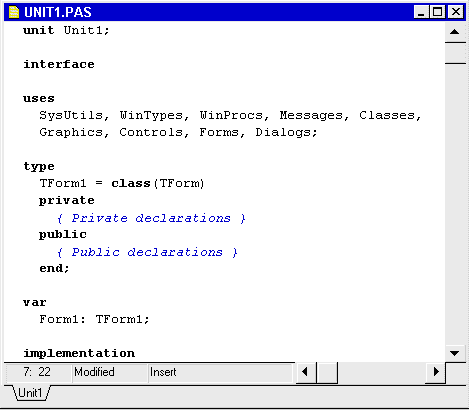


Рисунок 2. В окне Редактора описывается логика управления программой.

Дизайнер Форм в Delphi столь интуитивно понятен и прост в использовании, что создание визуального интерфейса превращается в детскую игру. Дизайнер Форм первоначально состоит из одного пустого окна, которое заполняется всевозможными объектами, выбранными на Палитре Компонент.

Несмотря на всю важность Дизайнера Форм, местом, где программисты проводят основное время является Редактор. Логика является движущей силой программы и Редактор - то место, где она "кодируется".

Палитра Компонент (рис.3) позволяет выбрать нужные объекты для размещения их на Дизайнере Форм. Для использования Палитры Компонент просто первый раз необходимо щелкнуть мышкой на один из объектов и потом второй раз - на Дизайнере Форм. Выбранный объект появится на проектируемом окне и им можно манипулировать с помощью мыши.

Палитра Компонент использует постраничную группировку объектов. Внизу Палитры находится набор закладок - Standard, Additional, Dialogs и т. д. Если щелкнуть мышью на одну из закладок, то можно перейти на следующую страницу Палитры Компонент. Принцип разбиения на страницы широко используется в среде программирования Delphi и его легко можно использовать в своей программе. (На странице Additional есть компоненты для организации страниц с закладками сверху и снизу).



Рисунок 3. Палитра Компонент - место, где выбираются объекты, которые будут помещены на форму.

После помещения компонента TEdit на форму; можно двигать его с места на место. Также можно использовать границу, прорисованную вокруг объекта для изменения его размеров. Большинством других компонент можно манипулировать тем же образом. Однако, невидимые во время выполнения программы компоненты (типа TMenu или TDataBase) не меняют своей формы.

Слева от Дизайнера Форм можно видеть Инспектор Объектов (рис.4). Информация в Инспекторе Объектов меняется в зависимости от объекта, выбранного на форме. Важно понять, что каждый компонент является настоящим объектом и можно менять его вид и поведение с помощью Инспектора Объектов.

Инспектор Объектов состоит из двух страниц, каждую из которых можно использовать для определения поведения данного компонента. Первая страница - это список свойств, вторая - список событий. Если нужно изменить что-нибудь, связанное с определенным компонентом, то обычно делается это в Инспекторе Объектов. К примеру, можно изменить имя и размер компонента TLabel изменяя свойства Caption, Left, Top, Height, и Width.

Можете использовать закладки внизу Инспектора Объектов для переключения между страницами свойств и событий.

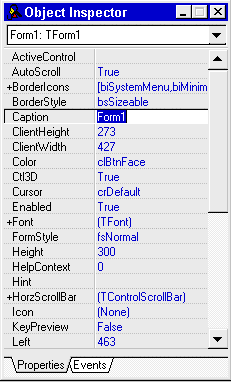


Рисунок 4: Инспектор Объектов позволяет определять свойства и поведение объектов, помещенных на форму.

Страница событий связана с Редактором; если дважды щелкнуть мышкой на правую сторону какого-нибудь пункта, то соответствующий данному событию код автоматически запишется в Редактор, сам Редактор немедленно получит фокус, и сразу же появляется возможность добавить код обработчика данного события.

Последняя важная часть среды Delphi - Справочник (on-line help). Для доступа к этому инструменту нужно просто выбрать в системном меню пункт Help и затем Contents. На экране появится Справочник, показанный на рис.5

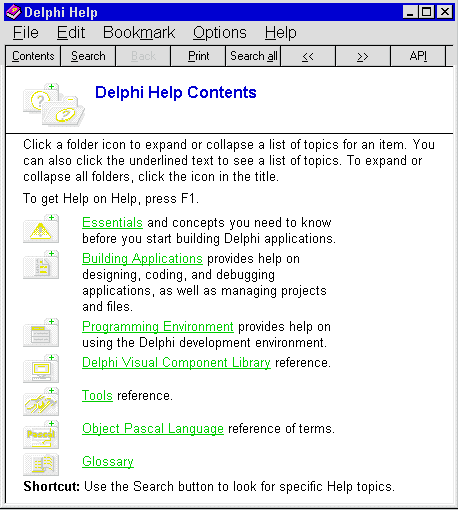


Рисунок 5: Справочник - быстрый поиск любой информации.

Справочник является контекстно-зависимым; при нажатии клавиши F1, появится подсказка, соответствующая текущей ситуации. Например, если находясь в Инспекторе Объектов, выберать какое-нибудь свойство и нажать F1 - откроется справка о назначении данного свойства. Если в любой момент работы в среде Delphi возникает неясность или затруднение – можно нажать F1 и необходимая информация появится на экране.

**43 Опишите алгоритм создания мультипликации.**

Большинство современных программ, работающих в среде Windows, являются мультимедийными. Такие программы обеспечивают просмотр видеороликов и мультипликации, воспроизведение музыки, речи, звуковых эффектов. Типичными примерами мультимедийных программ являются игры и обучающие программы.

Delphi предоставляет в распоряжение программиста два компонента, которые позволяют разрабатывать мультимедийные программы:

* Animate — обеспечивает вывод простой анимации (подобной той, которую видит пользователь во время копирования файлов);
* MediaPlayer — позволяет решать более сложные задачи, например, воспроизводить видеоролики, звук, сопровождаемую звуком анимацию.

**Компонент Animate**

Компонент Animate, значок которого находится на вкладке **Win32**(рис. 6), позволяет воспроизводить простую анимацию, кадры которой находятся в AVI-файле.

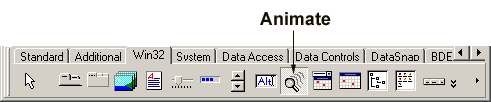


Рисунок 6.Значок компонента Animate

Хотя анимация, находящаяся в AVI-файле может сопровождаться звуковыми эффектами (так ли это — можно проверить, например, при помощи стандартной программы Проигрыватель Windows Media), компонент Animate обеспечивает воспроизведение только изображения. Для полноценного воспроизведения сопровождаемой звуком анимации следует использовать компонент меdiaPlayer.

Компонент Animate добавляется к форме обычным образом. После добавления компонента к форме следует установить значения его свойств. Свойства компонента Animate перечислены в таблице.

Свойства компонента Animate

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Свойство** | **Определяет** |  |
|  | Name | Имя компонента. Используется для доступа к свойствам компонента и управлением его поведением |  |
|  | FileName | Имя AVI-файла в котором находится анимация, отображаемая при помощи компонента |  |
|  | StartFrame | Номер кадра, с которого начинается отображение анимации |  |
|  | stopFrame | Номер кадра, на котором заканчивается отображение анимации |  |
|  | Activate | Признак активизации процесса отображения кадров анимации |  |
|  | Color | Цвет фона компонента (цвет "экрана"), на котором воспроизводится анимация |  |
|  | Transparent | Режим использования "прозрачного" цвета при отображении анимации |  |
|  | Repetitions | Количество повторов отображения анимации |  |

Следует еще раз обратить внимание, что компонент Animate предназначен для воспроизведения AVI-файлов, которые содержат только анимацию. При попытке присвоить записать в свойство FileName имя файла, который содержит звук, Delphi выводит сообщение о невозможности открытия указанного файла **(Cannot open**AVI). Чтобы увидеть, что находиться в AVI-файле: анимация и звук или только анимация, нужно из Windows раскрыть нужную папку, выделить AVI-файл и из контекстного меню выбрать команду **Свойства.**В результате этого откроется окно **Свойства,**на вкладке **Сводка**(рис. 7) которого будет выведена подробная информация о содержимом выбранного файла.

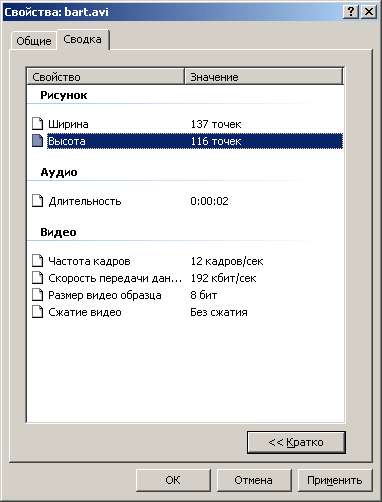


Рисунок 7. На вкладке **Сводка**отражается информация об AVI-файле

Процесс непрерывного воспроизведения анимации инициирует присваивоение значение True свойству Active.

Компонент Animate так же позволяет программисту использовать в своих программах стандартные анимации Windows. Вид анимации определяется значением свойства СommonAVI. Значение свойства задается при помощи именованной константы. В таблице приведены некоторые значения констант, вид анимации и описание процесса, для иллюстрации которого используется эти анимации.

Значение свойства comonAVi определяет анимацию

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Значение** | **Анимация** | **Процесс** |  |
|  | aviCopyFiles |  | Копирование файлов |  |
|  | AviDeleteFile |  | Удаление файла |  |
|  | aviRecycleFile |  | Удаление файла в корзину |  |

**Компонент MediaPlayer**

Компонент MediaPlayer, значок которого находится на вкладке **System**(рис. 8), позволяет воспроизводить видеоролики, звук и сопровождаемую звуком анимацию.

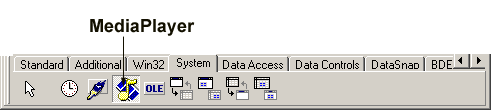


Рисунок 8.Значок компонента MediaPlayer

В результате добавления к форме компонента MediaPlayer на форме появляется группа кнопок (рис. 9), подобных тем, которые можно видеть на обычном аудио- или видеоплеере. Назначение этих кнопок и свойства компонента MediaPlayer пояснено в таблицах.

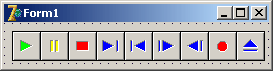


Рисунок 9.Компонент MediaPlayer

Кнопки компонента MediaPlayer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Кнопка** | **Обозначение** | **Действие** |  |
|  | Воспроизведение | btPlay | Воспроизведение звука или видео |  |
|  | Пауза | btPause | Приостановка воспроизведения |  |
|  | Стоп | btStop | Остановка воспроизведения |  |
|  | Следующий | btNext | Переход к следующему кадру |  |
|  | Предыдущий | btPrev | Переход к предыдущему кадру |  |
|  | Шаг | btStep | Переход к следующему звуковому фрагменту, например, к следующей песне на CD |  |
|  | Назад | btBack | Переход к предыдущему звуковому фрагменту, например, к предыдущей песне на CD |  |
|  | Запись | btRecord | Запись |  |
|  | Открыть/Закрыть | btEject | Открытие или закрытие CD-дисковода компьютера |  |

 Свойства компонента MediaPiayer

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Свойство** | **Описание** |  |
|  | Name  DeviceType  FileName  AutoOpen  Display  VisibleButtons | Имя компонента. Используется для доступа к свойствам компонента и управлением работой плеера  Тип устройства. Определяет конкретное устройство, которое представляет собой компонент MediaPiayer. Тип устройства задается именованной константой: dtAutoSelect — тип устройства определяется автоматически; dtVaweAudio — проигрыватель звука; dtAVivideo — видеопроигрыватель; dtCDAudio — CD-проигрыватель  Имя файла, в котором находится воспроизводимый звуковой фрагмент или видеоролик  Признак автоматического открытия сразу после запуска программы, файла видеоролика или звукового фрагмента  Определяет компонент, на поверхности которого воспроизводится видеоролик (обычно в качестве экрана для отображения видео используют компонент Panel)  Составное свойство. Определяет видимые кнопки компонента. Позволяет сделать невидимыми некоторые кнопки |  |

**77 Опишите способы представления графов в памяти компьютера.**

Граф - множество V вершин и набор E неупорядоченных и упорядоченных пар вершин. Говоря проще, граф — это множество точек (вершин) и соединяющих их путей (дуг или рёбер). Граф может быть ориентированным и неориентированным. В ориентированном графе пути (дуги) имеют направление, а в неориентированном - не имеют. Пути в неориентированном графе называются рёбрами.

Взвешенным называется такой граф, для каждого ребра(дуги) которого определена некоторая функция. А эта функция называется весовой.

Рассмотрим пример ориентированного взвешенного графа. На рис. 10 точками обозначены вершины графа, стрелками - дуги, чёрные числа - номера вершин графа, а красные - веса дуг. Вес дуги можно представить также как стоимость.

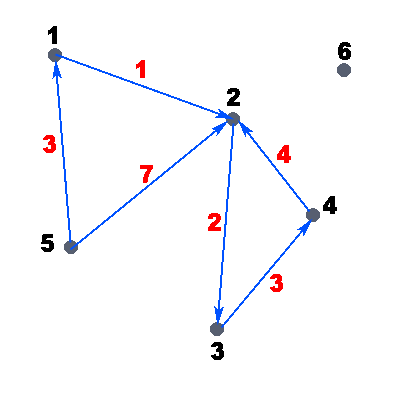


Рисунок 10. Пример графа.

Т. е. например: дан граф, нужно дойти от вершины i до вершины j, заплатив минимальное количество денег/потратив наименьшее количество времени. Пусть в нашем графе, который представлен на рисунке, нам нужно пройти из вершины 5 в вершину 2, а вес дуг - стоимость прохода по данному ребру. В данном примере очевидно, что выгоднее пройти через вершину 1 и только потом прийти в вершину 2, так мы заплатим всего 4 единицы денег, иначе, если пойти напрямую, мы заплатим целых 7 единиц.

Проблема правильного хранения графа в памяти компьютера действительно актуальна в сегодняшние дни. Рассмотрим основные способы хранения графов.

### **Матрица смежности**

Один из самых распространённых способов хранения графа - матрица смежности. Она представляет из себя двумерный массив. Если в клетке i,j установлено значение ПУСТО, то дуги, начинающейся в вершине i и кончающейся в вершине j, нет. Иначе дуга есть. Чаще всего за значение ПУСТО берут 0, а в клетки, которые обозначают наличие дуги, вписывают вес этой дуги. Если граф не взвешенный, то вес дуги считается равным единице. Нарисуем матрицу смежности для предыдущего графа:

|  | **1** | **2** | **3** | **4** | **5** | **6** |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | 0 | 1 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 2 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 3 | 0 | 0 |
| **4** | 0 | 4 | 0 | 0 | 0 | 0 |
| **5** | 3 | 7 | 0 | 0 | 0 | 0 |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 |

Употреблено слово “дуга”, т. к. матрица смежности приспособлена только для ориентированных графов. Очень много способов хранения приспособлены только для ориентированных графов. Однако почти во всех задачах ребро можно заменить двумя дугами, т. е. если у нас есть ребро (1,3), то мы можем заменить его на дуги (1,3) и (3,1) - так мы сможем пройти в любом направлении в любое время.

В матрице смежности часто нужно хранить большое количество нулей. Например, в матрице “нужных” значений только 6, а остальные 30 - нули, не представляющие почти никакой нужной информации.

Для представления графа матрицей смежности нужно V в квадрате (где V - количество вершин). Если граф почти полный, т. е. Е почти равно V в квадрате (где Е - количество дуг), этот способ хранения графа наиболее выгоден, т. к. его очень просто реализовывать и память будет заполнена “нужными” значениями.

Кроме большого количества требуемой памяти и медленной работы на разреженных графах у матрицы смежности есть ещё один важный недостаток. Иногда в задачах нужно выводить не номера вершин, а номера дуг(рёбер) на вводе. Хранить эти номера матрица смежности не умеет. Нужно реализовывать восстановление номера дуги(ребра) как-то иначе, а это совсем неудобно.

Временные сложности сведены в таблицу:

| **Операция** | **Временная сложность** |
| --- | --- |
| Проверка смежности вершин x и y | О(1) |
| Перечисление всех вершин смежных с x | О(V) |
| Определение веса ребра (x, y) | О(1) |
| Перечисление всех ребер (x, y) | О(V2) |

### **Описание Бержа**

Для того чтобы ускорить работу матрицы смежности на разреженных графах, можно использовать другой тип хранения графа - описание Бержа. Для каждой вершины хранится список смежных вершин. Чаще всего для этого используется двумерный массив размером V в квадрате, в строке u которого хранятся номера вершин, смежных с u. В нулевой элемент строки u вписывается количество вершин, хранящихся в данной строке. Теперь попробуем представить граф при помощи описания Бержа на примере:

|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 1 | 2 | 0 | 0 | 0 | 0 | 0 |
| **2** | 1 | 3 | 0 | 0 | 0 | 0 | 0 |
| **3** | 1 | 4 | 0 | 0 | 0 | 0 | 0 |
| **4** | 1 | 2 | 0 | 0 | 0 | 0 | 0 |
| **5** | 2 | 1 | 2 | 0 | 0 | 0 | 0 |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

В нулевом столбце хранятся “длины” строк. Однако, вес дуг никак не хранится, а если его хранить отдельно, то нужно заводить ещё один массив размером V в квадрате…

Временные сложности сведены в таблицу:

| **Операция** | **Временная сложность** |
| --- | --- |
| Проверка смежности вершин x и y | О(V) |
| Перечисление всех вершин смежных с x | О(V) |
| Определение веса ребра (x, y) | Вес не хранится |
| Перечисление всех ребер (x, y) | О(Е) |

### **Список дуг**

Следующий тип хранения графа в памяти компьютера - список дуг. Чаще всего это двумерный массив размером 3\*E, в первой строке которого хранится информация, из какой вершины начинается дуга, во второй - в какой кончается, а в третьей строке - вес дуги. Опять же разберёмся на примере:

|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 1 | 2 | 3 | 4 | 5 | 5 | 0 | 0 |
| **2** | 2 | 3 | 4 | 2 | 1 | 2 | 0 | 0 |
| **3** | 1 | 2 | 3 | 4 | 3 | 7 | 0 | 0 |

Почти вся таблица заполнена “нужными” значениями, а не нулями. Это уже хорошо, значит, память экономится.

Временные сложности сведены в таблицу:

| **Операция** | **Временная сложность** |
| --- | --- |
| Проверка смежности вершин x и y | О(E) |
| Перечисление всех вершин смежных с x | О(E) |
| Определение веса ребра (x, y) | О(E) |
| Перечисление всех ребер (x, y) | О(E) |
| Поиск i-ой дуги | О(1) |

Как видно, этот способ, в отличие от матрицы смежности и описания Бержа, хранит информацию о номере дуги. Также ясно, что этот способ нам выгоден, если чаще всего нам нужно будет узнать что-то (вес, вершины начала или конца) о i-ой дуге. Однако, такие задачи в практическом программировании встречаются довольно редко.

Если в предыдущих представлениях одно ребро мы заменяли двумя дугами, то список дуг может хранить или дуги или рёбра (в зависимости от реализации). Это довольно удобно и может требовать в 2 раза меньше памяти.

### **Список смежности**

Он представляет из себя два массива: vert и adj.

Из каждой вершины может выходить 0, 1 и более дуг, причём вершин будет V или менее. Если из вершины i не выходит дуг (т. е. количество дуг равно нулю), то в массиве vert в i-ой клетке будет храниться значение ПУСТО. Если из вершины i выходит хотя бы одна дуга, то в массиве vert в i-ой клетке будет храниться номер элемента массива adj, в котором хранится конечная вершина дуги. Также в массиве adj хранится вес дуги, и указатель на следующую “конечную” вершину дуги, которая начинается в вершине i. Поначалу может показаться, что этот способ очень запутан, т. к. один массив указывает на другой, другой сам на себя, при чём много раз. Однако это не совсем так, т. к. следует лишь несколько раз самому попробовать реализовать хранение графа таким способом, и он кажется очень мощным, полезным и несложным. Давайте сами попробуем разобраться на примере.

На вход подаются следующие данные:

6 6

5 1 3

1 2 1

4 2 4

2 3 2

3 4 3

5 2 7

Так выглядит массив vert:

| **1** | **2** | **3** | **4** | **5** | **6** |
| --- | --- | --- | --- | --- | --- |
| 2 | 4 | 5 | 3 | 1 | 0 |

А вот так массив adj:

|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 1 | 2 | 2 | 3 | 4 | 2 | 0 | 0 |
| **2** | 3 | 1 | 4 | 2 | 3 | 7 | 0 | 0 |
| **3** | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

В массиве adj первая строка - конечная вершина дуги, вторая - её вес, а третья - ссылка на следующую.

В этом примере дуги введены не по порядку, чтобы можно было лучше разобраться, и было видно, что позиции могут быть совершенно различными. Например, первая конечная вершина дуги, начинающейся в вершине 5, лежит в массиве adj на первой позиции, а не на пятой.

Рассмотрим временные сложности:

| **Операция** | **Временная сложность** |
| --- | --- |
| Проверка смежности вершин x и y | О(E) |
| Перечисление всех вершин смежных с x | О(E) |
| Определение веса ребра (x, y) | О(E) |
| Перечисление всех ребер (x, y) | О(E) |
| Поиск i-ой дуги | О(1) |

Времена у всех операций, вроде бы, такие же, как и у списка рёбер. Однако, большинство из них реально намного меньше. Например, проверка смежности вершин x и y и перечисление всех вершин смежных с x в списке рёбер гарантированно будет выполнятся О(Е) операций, т. к. обязательно нужно пробежать весь список, а в списке смежности мы бежим только по вершинам, смежным с х.

### **Оптимизация к списку смежности**

Для того, чтобы добавить дугу (i,j) мы должны начать из вершины i и пройти по всем вершинам, смежными с нею. И так каждый раз! А что если вершине х инцидентны E рёбер? Тогда сложность добавления O(E в квадрате)!

А почему бы не добавлять ребро сразу туда, куда надо? Конечно, при считывании, сразу записываются данные в нужную ячейку (в конец получившегося списка). Но ведь на эти данные нужно сделать указатель. Самое тяжёлое - найти место, где делать указатель. Поиск этого места и занимает основу драгоценного времени. А что, если хранить указатель не только на первую конечную вершину дуги, но и на последнюю? Тогда проблема решена. Добавление занимает О(1) времени, но требует дополнительно О(V) памяти…

Рассмотрим данную оптимизацию на примере. Изменится только вид массива vert, поэтому рассмотрим только его:

|  | **1** | **2** | **3** | **4** | **5** | **6** |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | 2 | 4 | 5 | 3 | 1 | 0 |
| **2** | 2 | 4 | 5 | 3 | 6 | 0 |

Здесь в первой строке массива vert хранится указатель на первую конечную вершину, во второй - на последнюю. В данном примере польза от этой оптимизации почти не заметна, т.к. максимальное вершин, исходящих из одной дуги, равно 2, что очень мало. Если дуг, инцидентных одной вершине, будет много, то и ускорение будет довольно большое и “лишняя” выделенная память окупится.

### **Заключение**

Универсального способа хранения графа в памяти компьютера, который бы хранил максимально много информации о графе, работал с любыми графами и работал бы быстро нет. Для различных задач и графов оптимальны различные представления. Конечно, наиболее мощным и универсальным способом хранения является список смежности. С оптимизацией он работает быстрее и всё ещё требует не так уж и много памяти. Однако, если количество дуг приближается к квадрату количества вершин, то лучше матрицы смежности способа не найти, т. к. тогда скорость работы и количество памяти даже списка смежности приближается к матрице смежности, если не требует больше… Конечно, можно написать и список смежности, однако пишется он дольше матрицы смежности, да и работать с матрицей смежности намного проще.

Конечно, были рассмотрены не все способы представления графов в памяти компьютера. Рассмотрены лишь те, которые пишутся наиболее быстро, быстро работают, и требуют немного памяти. Рассмотрены только статические способы представления графа в памяти компьютера. Есть ещё различные динамические способы хранения графов, они занимают меньше памяти, иногда быстрее работают. Однако и пишутся они намного сложнее.

**91 Введите три числа. В зависимости от выбранного переключателя RadioButton вычислить сумму или произведение чисел.**

**РЕШЕНИЕ:**

Разместим на форме 2 компонента RadioButton, 3 Edit, 4 Label и Button и зададим им свойства Caption и у компонентов Edit выставим свойство Text в пустую строку. Внешний вид формы в процессе разработке приведен на рис. 11.

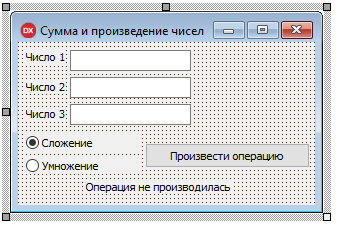


Рисунок 11. Форма в процессе разработки.

Зададим обработчик нажатия кнопки выполнения операции Button1.

procedure TForm1.Button1Click(Sender: TObject);

var

Res:single;

begin

if RadioButton1.Checked then

begin *{Выбрано сложение чисел}*

Res:=strtofloat(Edit1.Text)+strtofloat(Edit2.Text)+strtofloat(Edit3.Text);

*{Выводим результат операции в Label4}*

Label4.Caption:='Результат: '+Floattostr(Res);

end;

if RadioButton2.Checked then

begin*{Выбрано умножение чисел}*

Res:=strtofloat(Edit1.Text)\*strtofloat(Edit2.Text)\*strtofloat(Edit3.Text);

*{Выводим результат операции в Label4}*

Label4.Caption:='Результат: '+Floattostr(Res);

end;

end;

**ТЕСТ:**

Готовая к вводу данных форма представлена на рис. 12.

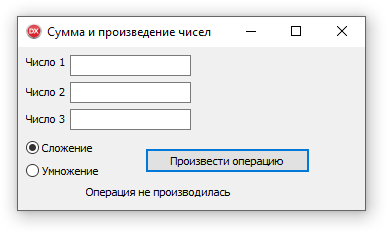


Рисунок 12. Приложение в режиме ожидания ввода данных.

Проверка сложения целых чисел представлена на рис. 13.

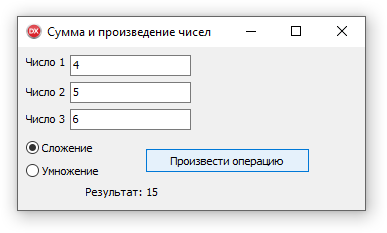


Рисунок 13. Приложение после сложения целых чисел.

Проверка умножения действительных чисел представлена на рис. 14.

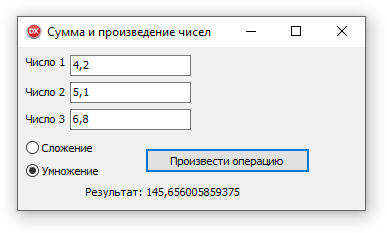


Рисунок 14. Приложение после перемножения действительных чисел.

**138 Создайте приложение, содержащее: список значений; поле для ввода новой строки и кнопку для добавления новой строки в список; поле со счетчиком для ввода номера удаляемой строки и кнопку для удаления выбранной строки. Предусмотреть соответствие вводимого номера и количества строк.**

**РЕШЕНИЕ:**

Разместим на форме компоненты Edit, Label, Button, UpDown, ListBox и зададим им свойства Caption и у компонентов Edit выставим свойство Text в пустую строку. Зададим 3 тестовых значения для компонента ListBox, свяжем компонент UpDown с компонентом Edit2.

Внешний вид формы в процессе разработке приведен на рис. 15.

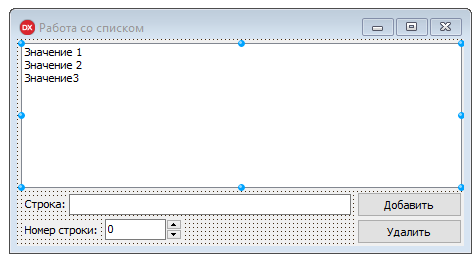
****

Рисунок 15. Проектирование формы приложения.

Создадим обработчик нажатия кнопки "Добавить". Нам не наложено никаких ограничений на содержимое строки для добавления, поэтому строка добавляется в конец списка без изменений, при попытке вставки пустой строки выводится сообщение.

procedure TForm1.Button1Click(Sender: TObject);

begin

*{проверка пустой строки}*

if EDit1.Text='' then

showMessage('Нельзя добавлять пустую строку')

else

*{добавление строки в конец списка}*

Listbox1.Items.Add(EDit1.Text);

end;

Создадим обработчик нажатия кнопки «Удалить» с проверкой номера строки на соответствие номеру существующей строки в компоненте ListBox. Нет нужды проверять отрицательные значения, так как задано свойство Min компонента UpDown равным нулю.

procedure TForm1.Button2Click(Sender: TObject);

var

n:integer;

begin

*{преобразуем строку в число}*

n:=strtoint(edit2.Text);

*{Проверим, есть ли строка с таким порядковым номером}*

if (N>Listbox1.count)or(N<1) then

ShowMessage('Задано удаление несуществующей строки')

else

*{удаляем строку, нумерация в компоненте начинается с 0, учитываем этот факт}*

Listbox1.Items.Delete(N-1);

end;

**ТЕСТ:**

Добавление строки в список отражено на рис. 16-17.

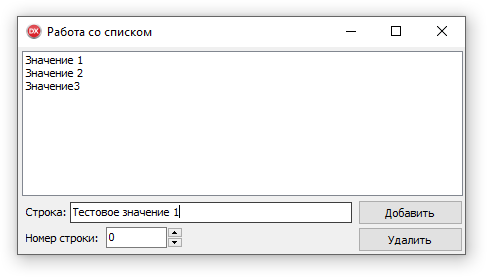


Рисунок 16. Ввод строки для добавления к списку.

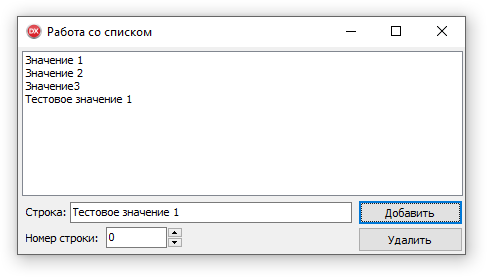


Рисунок 17. Результат операции добавления строки к списку.

Сообщение об ошибке добавления пустой строки к списку представлено на рис. 18.

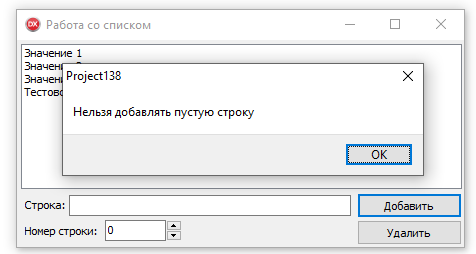


Рисунок 18. Реакция программы на добавление пустой строки.

Сообщение об ошибке строки с номером, выходящим за пределы номеров строк списка представлено на рис. 19.

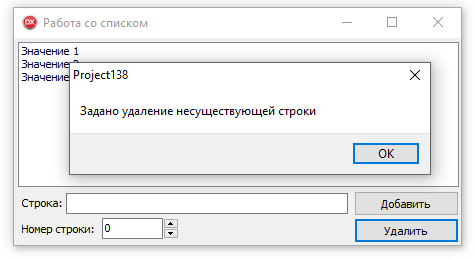


Рисунок 19. Удаление строки с несуществующим номером.

Результат удаления строки с номером 2 представлен на рис. 20.

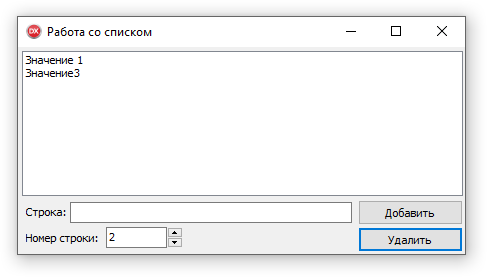


Рисунок 20. Результат удаления строки 2.

**159 Посчитайте число вершин в графе (граф задан матрицей смежности вершин), от которых отходит более m ребер. Указание. Найдите количество сумм строк в матрице смежности, больших m**.

**РЕШЕНИЕ:**

*Теоретическое обоснование способа решения:* матрица смежности представляет собой таблицу размера NxN где N общее количество вершин, элементами которой являются веса ребер (или 1 если вес не учитывается) либо 0 если ребра отсутствуют. Таким образом каждая строка(столбец) матрицы описывает одну вершину, а точнее ребра, отходящие от нее к другим вершинам (если эти ребра наличествуют). Таким образом, подсчитав сумму ненулевых элементов строки(столбца) таблицы получим количество ребер, которые отходят от данной вершины. Результатом и будет количество вершин с суммой ненулевых элементов более m.

Создадим форму приложения, разместим на ней необходимые компоненты. Для облегчения тестирования, отладки и проверки программы организуем загрузку данных из файла соответствующей кнопкой на форме рис.21. В папке проекта лежат тестовые файлы с именами вида 4vm2r1.txt, где 4 – количество вершин, 2 – значение m при котором 1 это правильный ответ. При загрузке файла значения m и n подставятся в поля автоматически.

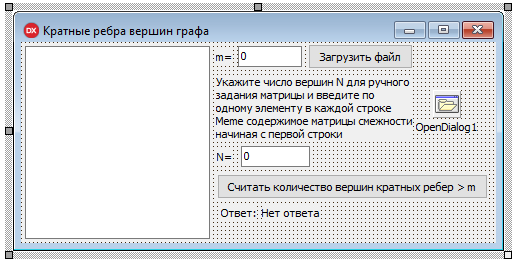


Рисунок 21. Разработка формы приложения.

Зададим обработчик кнопки загрузки данных из файла.

procedure TForm1.Button1Click(Sender: TObject);

begin

if OpenDialog1.Execute then

Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

end;

Зададим обработчик кнопки подсчета количества вершин.

procedure TForm1.Button2Click(Sender: TObject);

var

a:array[1..10,1..10] of integer;

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

s:string;

begin

Label5.caption:='Нет ответа';

if Memo1.Lines.Count<>strtoint(edit2.Text)\*strtoint(edit2.Text) then

begin

ShowMessage('Указано неверное количество вершин для матрицы');

end

else

begin

n:=0*;{Инициализируем переменные}*

c:=0;

for I := 1 to strtoint(edit2.Text) do

begin*{в цикле по каждой строке}*

k:=0; *{Инициализируем переменные}*

s:='';

for j := 1 to strtoint(edit2.Text) do

begin *{работаем со строкой перебирая столбцы}*

*{вносим элемент в матрицу}*

a[i,j]:=strtoint(memo1.Lines[c]);

*{увеличиваем счетчик ребер вершины}*

if a[i,j]>0 then k:=k+1;

*{вспомогательная строка для вывода матрицы}*

s:=s+' '+memo1.Lines[c];

c:=c+1;

end;

*{по условию увеличиваем счетчик вершин с кратными ребрами}*

if k>strtoint(edit1.Text) then n:=n+1;

Memo1.Lines.Add(s);

end;

end;

Label5.Caption:=inttostr(n);

end;

**ТЕСТ:**

Запущенное приложение представлено на рис.22.

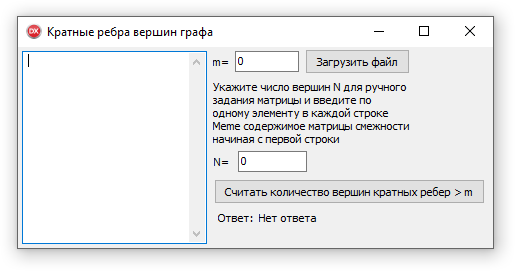
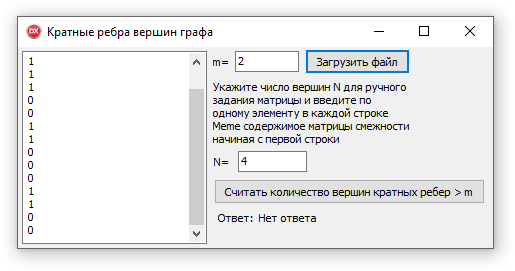


Рисунок 22. Запущенное приложение.

Приложение с введенными данными матрицы смежности с количеством вершин 4 и условием m=2 представлено на рис. 23.



Результат работы приложения представлен на рис. 24.

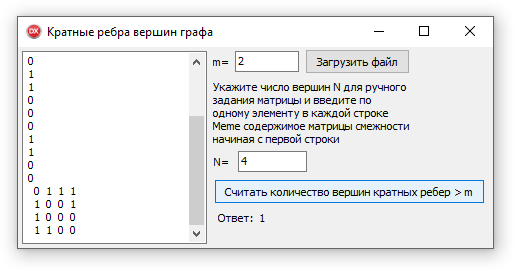


Рисунок 24. Результат работы приложения.

Результат работы приложения для 5 вершин с ребрами разного веса представлен на рис. 25.

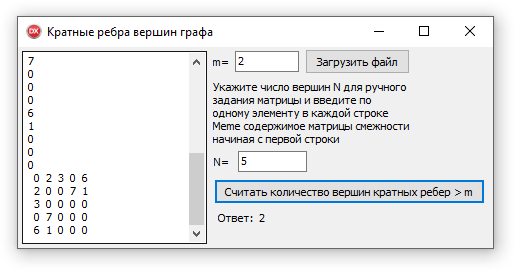


Рисунок 25. Результат работы приложения.

Результат работы приложения при неверно заданной матрице вершин из тестового файла 4vm2r1E.txt представлен на рис.26.

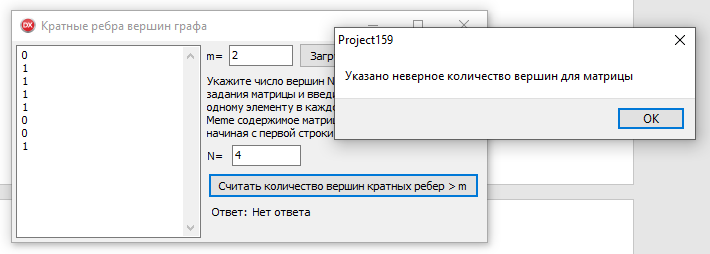


Рисунок 26. Результат обработки ошибки.

**Список использованных источников**

1. ГОСТ ИСО/ МЭК 2382-99. Информационные технологии. Словарь. Ч. 1. Основные термины.
2. ГОСТ19.701-90 (ИСО 5807-85). Описание символов. Правила применения символов и выполнения схем.
3. СТУ СМК 01-32-2019. Стандарт учреждения. Общие требования к оформлению текстовых документов».
4. Могилев, А.В. Методы программирования. Компьютерные вычисления / А.В. Могилев, Л.В. Листрова. - Санкт-Петербург: 2008.
5. Милованов, И.В. Основы разработки программного обеспечения вычислительных систем / И.В. Милованов, В.И. Лоскутов. – Тамбов: 2011.
6. Кибардин, А.В. Основы информатики / А.В. Кибардин. – Екатеринбург: 2015.
7. Цветков, А.С. Язык программирования PASCAL / А.С. Цветков.– Санкт-Петербург: 2015.
8. Макаров, В.Л. Программирование и основы алгоритмизации / В.Л. Макаров. – Санкт-Петербург: 2003.
9. Котов, В.М. Информатика. Методы алгоритмизации / В.М. Котов. – Нар. асвета: 2000.
10. Зубок, Д.А. Основы программирования в среде TURBO PASCAL / Д.А Зубок, С.В. Краснов, А.В. Маятин. – Санкт-Петербург: 2009.