**МИНИСТЕРСТВО ТРАНСПОРТА И КОММУНИКАЦИЙ
РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТРАНСПОРТА»**

Кафедра «Информационные технологии»

ИНФОРМАТИКА И КОМПЬЮТЕРНОЕ ПРОЕКТИРОВАНИЕ

Учебно-методическое пособие

Гомель 2015

МИНИСТЕРСТВО ТРАНСПОРТА И КОММУНИКАЦИЙ
РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТРАНСПОРТА»

Кафедра «Информационные технологии»

ИНФОРМАТИКА И КОМПЬЮТЕРНОЕ ПРОЕКТИРОВАНИЕ

Одобрено методической комиссией заочного факультета
в качестве учебно-методического пособия
для студентов специальности
1 37 02 05 «Строительство железных дорог, путь и путевое хозяйство»

Гомель 2015

УДК 004.415.2 (075.8)
ББК 32.81
И74

Авторы: *Е. Л. Миняйлова, Д. А. Вербовиков, Н. Р. Коледа, В. С. Миняйлов*

Рецензент – канд. техн. наук, доцент кафедры информационных технологий *О. П. Гораев* (УО «БелГУТ»)

Информатика и компьютерное проектирование : учеб.-метод. по-
И74 собие / Е. Л. Миняйлова [и др.]; М-во трансп. и коммуникаций Респ. Беларусь, Белорус. гос. ун-т трансп. – Гомель : БелГУТ, 2015. – 68 с.
ISBN 978-985-554-468-6

Раскрываются современные представления о поиске графической информации в сети Интернет, работе с изображениями в CorelDraw, способах проектирования и анализа алгоритмов в задачах маршрутизации как наиболее актуальных для студентов университета транспорта.

Задания и упражнения направлены на приобретение студентами навыков работы с инструментальными средствами программирования.

Предназначено для студентов заочного факультета специальности 1 37 02 05 «Строительство железных дорог, путь и путевое хозяйство».

УДК 004.415.2 (075.8)
ББК 32.81

ISBN 978-985-554-468-6

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Важнейшая особенность предлагаемого курса в том, что содержание разбивается на множество мелких тем. Количество материала в теме не превышает критический порог достижения быстрого понимания и перехода к программированию. Каждая тема имеет определенную смысловую нагрузку и выполняет одно из действий, предусмотренных исходным заданием.

Проектирование, целью которого является поиск функционально эффективных решений, возможно благодаря использованию вычислительной техники и методам, позволяющим быстро просчитывать многочисленные варианты промежуточных и конечных решений, на основе которых был разработан объект. В данном учебно-методическом пособии рассматривается класс задач о поиске кратчайших расстояний и путей как наиболее актуальный для студентов университета транспорта. Содержание пособия раскрывает современные представления о поиске графической информации в сети Интернет, работе с графическими изображениями в CorelDraw, базовых структурах данных, обмене информацией между приложениями, стилевом оформлении документа в текстовом процессоре.

Пособие предназначено для студентов, обучающихся по специальности 1 37 02 05 «Строительство железных дорог, путь и путевое хозяйство».

ЗАДАНИЕ НА КОНТРОЛЬНУЮ РАБОТУ

Исходные данные: карта железных дорог.

Содержание работы

Задание 1.

Используя информационные ресурсы сети Интернет, осуществить поиск карт железных дорог:

1) по ключевым словам. Например, «Карта железных дорог Беларуси» (рисунок 1), электронный ресурс: www.karty.by/wp-content/uploads/2014/11/x_railways.jpg;

2) по графической информации с помощью поисковых систем.

Рисунок 1 – Карта железных дорог Беларуси

Задание 2. С помощью графического редактора получить изображение графа с карты, аналогичное представленному на рисунке 2 фрагменту.

а) б)

Рисунок 2 – Карта железных дорог Беларуси:
а – фрагмент карты; *б* – граф фрагмента карты

На фрагменте карты и на графе указать номера вершин и веса рёбер. Вес ребра может обозначать длину ребра в километрах, время в пути и т.п.

Задание 3. Используя граф (задание 2), осуществить:

1) поиск пути; обход невзвешенного графа в глубину с помощью структуры данных «стек»;

2) поиск кратчайшего пути; обход невзвешенного графа в ширину с помощью структуры данных «очередь»;

3) поиск кратчайших путей на взвешенном графе из одной вершины во все остальные с помощью алгоритма Дейкстры;

4) поиск кратчайших путей на взвешенном графе между всеми парами вершин с помощью алгоритма Флойда.

Пункты 1–4 выполнить в электронной таблице. Проверить правильность вычислений с помощью программ, предоставленных преподавателем.

Сформировать итоговый отчет в текстовом процессоре.

1 СОВРЕМЕННЫЕ СРЕДСТВА ПОИСКА ГРАФИЧЕСКОЙ ИНФОРМАЦИИ В СЕТИ ИНТЕРНЕТ

Многие поисковые системы позволяют осуществлять поиск не только текстовой информации, но и графической.

Существующие поисковые системы ищут графические файлы, анализируют всю возможную информацию об объекте: учитывают текстовое описание, распознают графические образы, анализируют речь и другую звуковую информацию (если это видеофайл). Но в первую очередь поиск ведется по содержанию тегов на веб-странице и имени графического файла.

Поисковыми системами, с помощью которых можно найти графическую информацию являются: Google, Yahoo, Yandex, Mail, Rambler, AltaVista и др.

Также выделяют две специализированные технологии поиска, основанные на обработке визуальной информации:

1) не анализирующие изображение, а применяющие технологии поиска схожих по каким-либо параметрам объектов. Эти технологии используются ресурсами Taggalaxy.de, Like.com, Tineye.com, Picitup.com, Tiltomo.com;

2) анализирующие содержание изображения или видео. Используются ресурсами Blinkx.com, Betaface.com, Delvenetworks.com.

Поскольку разные поисковые сервисы используют неодинаковый подход к подбору ресурсов, соответствующих запросу, результаты поиска будут различные. Поэтому, если долгие поиски одним способом по известным сервисам ни к чему не привели, следует сменить тактику и попробовать другие способы поиска информации или альтернативные поисковые системы.

1.1 Поисковая система Google

Рассмотрим основные возможности поиска графической информации на примере поисковой системы Google.

Первый способ: чтобы попасть в раздел «Картинки», нужно выбрать вкладку с соответствующим названием, ввести в поисковую строку слово-запрос и нажать «Поиск картинок». Поисковая система Google помимо основного запроса выдаёт так называемые «ассоциации» (список запросов, которые часто подаются пользователями в течение одной поисковой сессии), с помощью которых система пытается уточнить ваш запрос (рисунок 3).

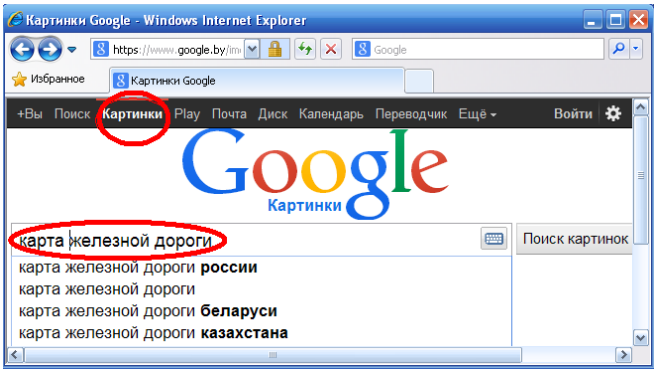


Рисунок 3 – Первый способ поиска графической информации

В результатах поиска отображаются уменьшенные копии найденных «картинок». Далее, воспользовавшись вкладкой «Инструменты поиска», можно с помощью расширенного меню поиска уточнить размер, цвет, тип картинки и время загрузки в виртуальную базу. Т. е. можно искать, например, только черно-белые изображения или только черно-белые фотографии.

Для реализации *второго способа* достаточно перетащить файл с изображением в значок фотоаппарата в поисковой строке и при необходимости дописать текст-описание. Как результат Google покажет, на каком сайте размещён исходный файл и какие есть ещё похожие изображения в сети Интернет (рисунок 4).

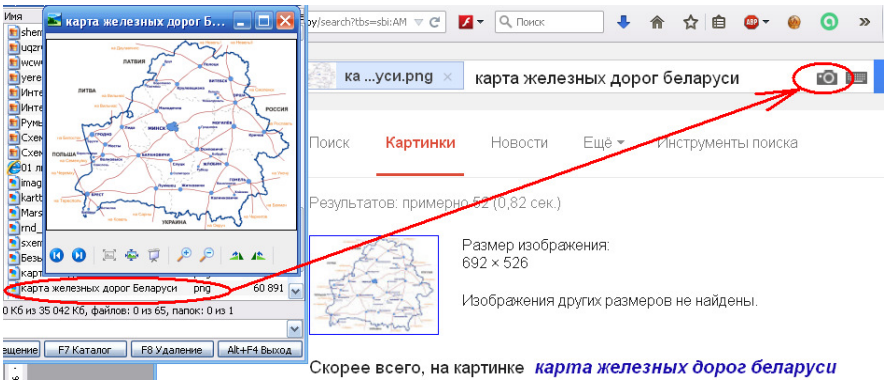


Рисунок 4 – Второй способ поиска графической информации

Третьим способом можно воспользоваться, скопировав ссылку на изображение и вставив ее в форму поиска по картинке (рисунки 5–7). Данная форма открывается после клика по значку фотокамеры. Поисковая система найдёт файлы, похожие на запрашиваемый оригинал как по содержанию, так и по цвету.

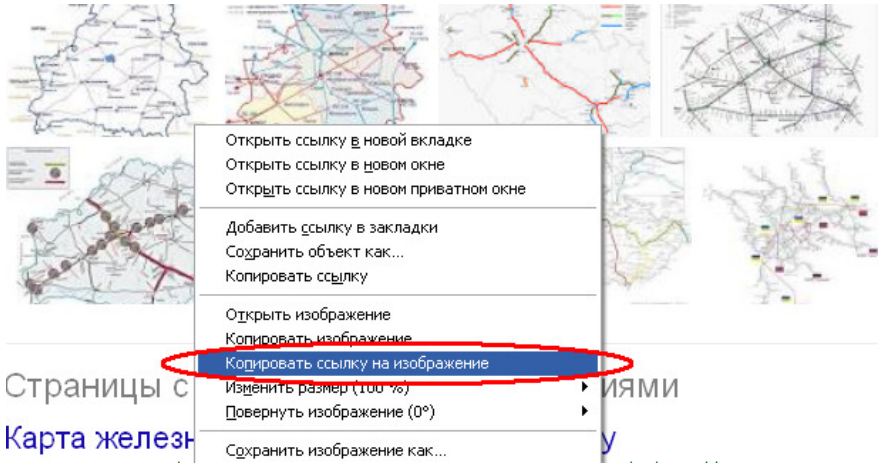


Рисунок 5 – Третий способ поиска графической информации: шаг 1

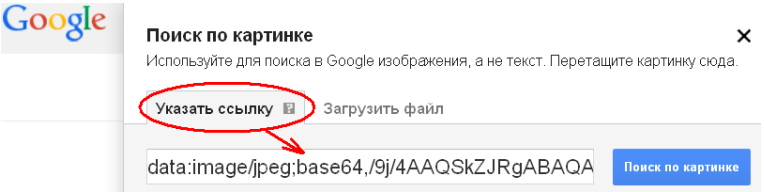


Рисунок 6 – Третий способ поиска графической информации: шаг 2

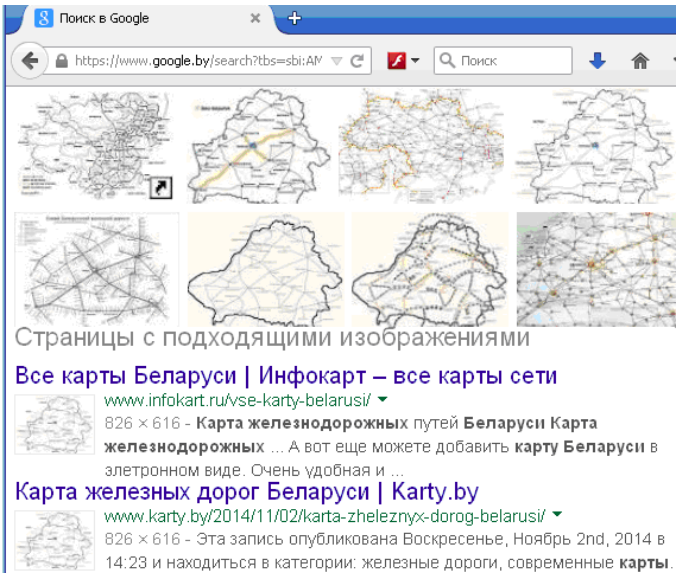


Рисунок 7 – Результаты поиска с помощью третьего способа

Четвертый способ основан на установке специального расширения в браузер, например, «Mozilla Firefox». После этого надо правой кнопкой мыши вызвать контекстное меню для картинки, затем выбрать команду «Search by image for Google» (рисунок 8). Google в новой вкладке откроет страницу и выполнит поиск похожих изображений.

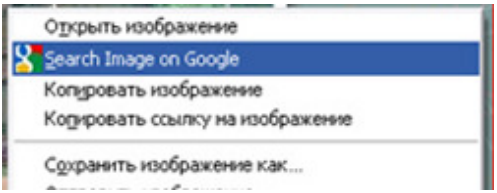


Рисунок 8 – Четвертый способ поиска графической информации

1.2 Сервис TinEye.com

Сервис TinEye.com ищет копии изображений в сети Интернет.

Для поиска можно загрузить файл с фотографией или же скопировать ссылку на изображение и вставить ее в форму поиска по картинке. TinEye.com произведёт поиск и представит список похожих изображений с указанием сайтов, на которых они размещены.

1.3 Сервисы Like.com и Picitup.com

Like.com осуществляет поиск по демонстрациям товаров в электронных или виртуальных магазинах (рисунок 9).

Рисунок 9 – Результаты поиска с помощью сервиса Like.com

Сервис Picitup.com работает наподобие Like.com. Поиск осуществляется на основе текста и мультимедиа-информации в рамках поиска похожих изображений без анализа их содержимого.

1.4 Сервис Tiltomo.com

Tiltomo.com действует как каталог, использующий в своей работе базу Flickr и производящий по требованию пользователя дополнительную фильтрацию выводимых фото (рисунок 10).

Рисунок 10 – Результаты предварительного поиска с помощью сервиса Tiltomo.com

1.5 Сервис для обмена графическими файлами Flickr.com и поисковая система Taggalaxy.de

Flickr.com является одним из первых Web 2.0 сервисов для обмена графическими файлами (рисунок 11).

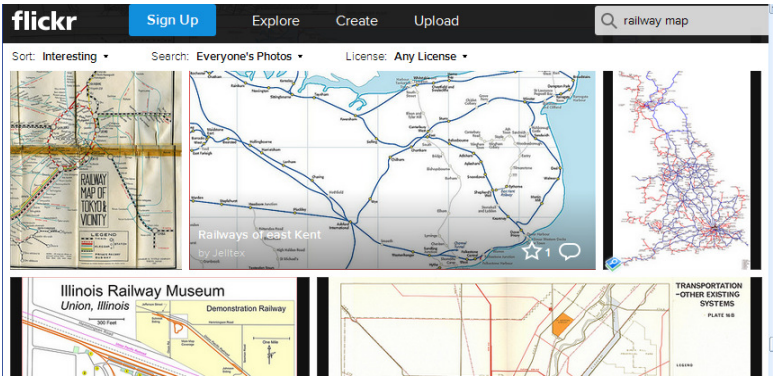


Рисунок 11 – Сервис для обмена графическими файлами Flickr.com

Так как на сервере этого сервиса очень много загружаемых картинок, то к нему необходима отдельная поисковая система.

Одним из необычных поисковиков является Taggalaxy.de. Этот поисковый сервис представляет собой средство с предварительным просмотром для поиска изображений на Flickr.com. Интерфейс его поиска графический (рисунок 12). Недостатком этой поисковой системы является то, что запросы можно вводить только латиницей (рисунок 13).

Рисунок 12 – Запрос «railway map»

Рисунок 13 – Результаты поиска по запросу «railway map»

Сайт Taggalaxy.de – визуально привлекательный поисковик, который осуществляет поиск изображений на сервисе Flickr по тэгам.

После того как будет выполнен запрос по ключевому слову, на экране возникнет система из «Солнца» и «планет», которые вращаются вокруг све-

тила. Каждое небесное тело имеет свое предназначение и «подписано» словом. В центре «галактики» – ключевой запрос, все остальные тела – это вспомогательные слова, уточнения. Полученную трехмерную модель с фотографиями можно поворачивать в виртуальном пространстве, подробно рассматривая и разыскивая нужное изображение.

Контрольные вопросы

1 Поиск графической информации в сети Интернет.

2 Основные особенности поиска графической информации на примере поисковой системы Google.

3 Поиск графической информации в сети Интернет с помощью сервисов.

2 ОБРАБОТКА ИЗОБРАЖЕНИЙ В ВЕКТОРНОМ ГРАФИЧЕСКОМ РЕДАКТОРЕ CORELDRAW

2.1 Технология рисования

1 Откройте приложение CorelDraw: Пуск / Все программы / **CorelDRAW Graphics Suite X4 / CorelDRAW X4**.

Создайте новый файл. Настройте размеры и ориентацию страницы рисования. Сохраните файл.

В дальнейшем не забывайте периодически сохранять сделанную работу для предотвращения потери данных из-за неустойчивой работы приложения или оборудования.

2 Инструментом **Эллипс** нарисуйте окружность (рисунок 14).

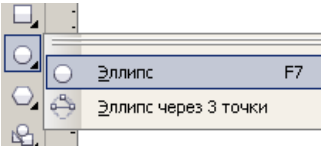


Рисунок 14 – Вызов инструмента Эллипс на панели рабочего инструментария

Правильная фигура рисуется при нажатой клавише Ctrl или настройкой одинакового размера по горизонтали и по вертикали (рисунок 15).

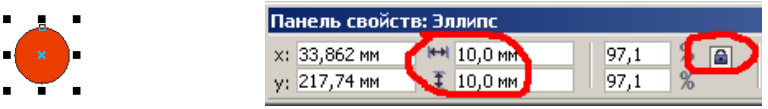


Рисунок 15 – Настройка одинакового габаритного размера фигуры

3 Впишите в окружность число. Для этого надо выбрать инструмент **Текст**  . Подведите курсор вверх к контуру. После этого курсор

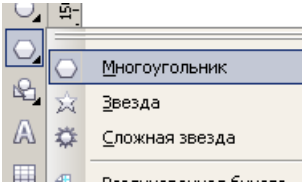
изменится на курсор ввода текста . После щелчка можно печатать текст внутри контура.

Сразу после набора числа необходимо применить команду **Текст►Простой текст►Текст в рамку**, чтобы число заняло весь контейнер.

Изменить числовое значение можно двойным щелчком инструмента **Текст** по контейнеру и заменить выделенное число на новое.

4 Инструментом **Многоугольник** нарисуйте правильный восьмиугольник (рисунок 16).

Рисунок 16 – Вызов инструмента Многоугольник



Правильная фигура рисуется при нажатой клавише Ctrl или настройкой равного габаритного размера по горизонтали и по вертикали (рисунок 17).

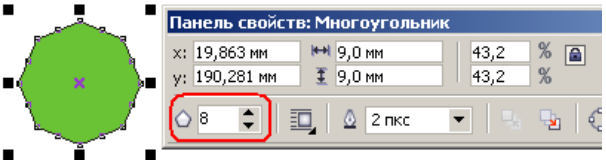


Рисунок 17 – Настройка равного габаритного размера фигуры

5 Впишите в многоугольник некоторое число, например, 84 (рисунок 18).

Рисунок 18 – Работа с инструментом **Текст**

6 Соедините окружность и восьмиугольник инструментом **Соединительная линия** . Для этого надо выбрать тип линии **Прямая соединительная линия** и соединять **узлы** на контурах объектов (рисунок 19).

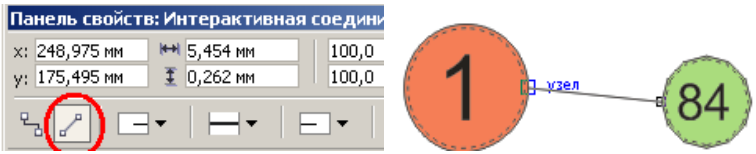


Рисунок 19 – Работа с инструментом Соединительная линия

Однако при соединении объектов оказывается, что соединить окружность и восьмиугольник аккуратно не получается, не хватает удобно расположенных узлов. Исправить это можно, изменив шаблон окружности. Для этого надо:

а) выбрать инструмент **Указатель** и выделить шаблон;

б) применить команду **Упорядочить►Преобразовать в кривую** или нажать кнопку  ;

в) выбрать инструмент **Форма**  , выделить им кривую и дважды щелкнуть инструментом **Форма** в том месте, где требуется добавить узел (рисунок 20). На окружности появится новый узел.

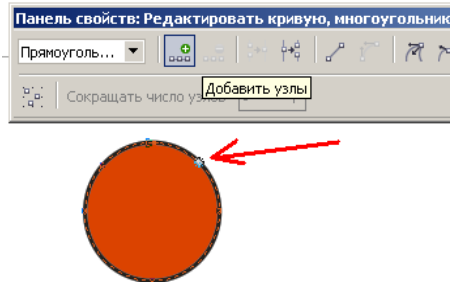


Рисунок 20 – Добавление нового узла на окружности

Нарисуйте соединительные линии. Небольшим перемещением вершин графа убедитесь, что соединительная линия проведена правильно, т. е. при перемещении соединённых объектов линия интерактивно реагирует на это изменение (рисунок 21).

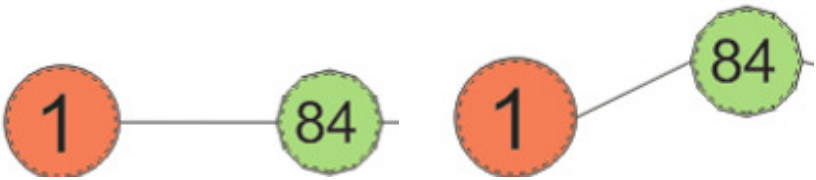
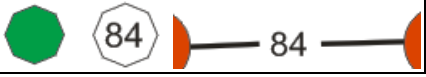


Рисунок 21 – Следование соединительной линии за перемещаемым объектом

2.2 Создание графа дорог по карте

Граф состоит из множества вершин и соединяющих их линий (ребер). Под графом дорог будем понимать векторное графическое изображение, созданное на основе карты железных дорог. Он создается по выбранным на исходной карте объектам дорожной сети.

Для создания графа по карте используйте объекты:

Элементы графа	Используемые объекты CorelDraw	Пояснения
Вершина	**Эллипс** 	Исходный образец – шаблон, а все остальные вершины – клоны.
Ребро	**Линия** 	Рисуется с помощью соединительной линии
Вес	**Текст** 	Рисуется с помощью 8-угольника с вписанным в него числом

Задача. Дана карта. Надо нарисовать изображение графа, соответствующее объектам на карте.

Решение.

1 Создайте новый файл. Настройте размеры и ориентацию страницы рисования. Сохраните файл.

2 Командой **Файл►Импорт** вставьте на страницу рисования растровое изображение с картой дорог, которую будете представлять графом.

3 Выровняйте растровое изображение по левому и верхнему краю командой **Упорядочить► Выровнять и распределить** (рисунок 22).

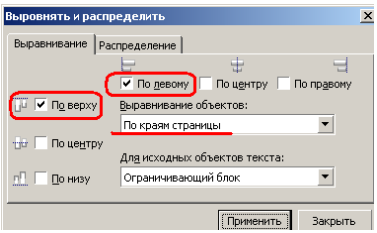


Рисунок 22 – Выравнивание изображения на странице

4 Затем карту надо сделать недоступной для смещения (команда **Упорядочить►Блокировать объект**).

Изображение карты железных дорог готово к использованию для рисования на нём графа.

5 На свободном от карты месте окна рисования инструментом **Эллипс** нарисуйте шаблон для вершины графа. Для лучшей видимости вершин на фоне карты назначьте шаблону яркую заливку (рисунок 23). По окончании размещения вершин графа эту заливку можно изменить одновременно для всех вершин-клонов.

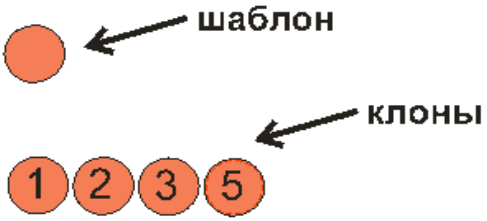


Рисунок 23 – Создание шаблона

6 Сразу же преобразуйте эллипс в кривую и добавьте на контур узлы.

7 Создайте клон исходного круга. Для этого выделите круг и примените команду **Правка►Клонирование.**

Этот круг будет шаблоном при создании клонов для вершин графа.

Справка: во время клонирования объекта создается копия объекта, которая связана с оригиналом. Любые изменения, выполненные для исходного объекта, автоматически распространяются и на клон. Однако изменения, выполненные для клона, не распространяются автоматически на исходный объект. Клонирование позволяет **одновременно** изменить несколько копий объекта, изменив объект шаблона.

Если в меню **Правка** отсутствует команда **Клонировать**, то ее надо добавить в состав меню. Для этого выберите команду **Инструменты►Настройка**, а затем в списке пункты **Команды►Правка** найдите пункт **Клонировать** и перетащите его мышью на название меню **Правка** в строке меню программы. Не отпуская кнопку мыши, поместите команду на желаемое место в меню. После освобождения кнопки мыши команда появится в меню на выбранном месте. Щелкните на кнопке **ОК** в диалоговом окне, чтобы завершить настройку меню.

8 Из полученного клона создайте достаточное количество клонов для вершин графа (Ctrl+D). Для этого выделите клон и нажмите несколько раз Ctrl+D (команда **Правка►Дублировать**).

Результат представлен на рисунке 24.

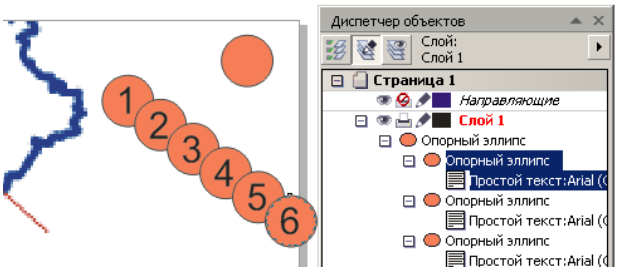
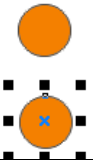
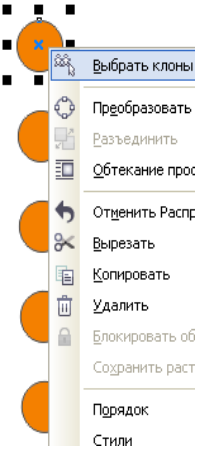
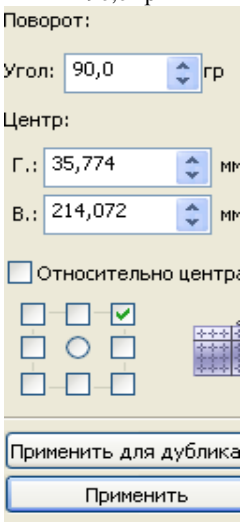
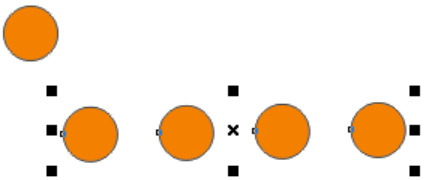
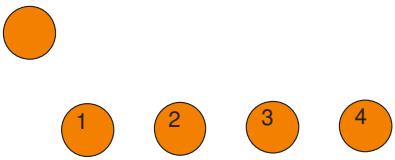
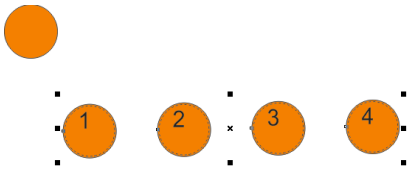


Рисунок 24 – Создание клонов шаблона

Проиллюстрируем операции данной работы более подробно:

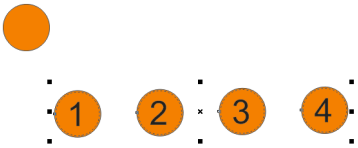
<table>
<tr>
<td>8.1 Нарисовать круг
  </td>
<td>8.2 Добавить цвет
  </td>
<td>8.3 Нажать Правка ► Клонирование
  </td>
<td>8.4 Сдвинуть клон вниз
  </td>
</tr>
<tr>
<td>8.5 Увеличить количество клонов Ctl+D
  </td>
<td>8.6 С помощью контекстного меню мыши выделить клоны
 
</td>
<td>8.7 Все клоны будут автоматически выделены
  </td>
<td>8.8 Развернуть клоны горизонтально: Упорядочить ► Преобразование ► Поворот. Установить Угол 90,0 гр
 
</td>
</tr>
<tr>
<td colspan="2">8.9 Получим горизонтальное размещение клонов
  </td>
<td colspan="2">8.10 Вписать числа в контейнеры-клоны
 
</td>
</tr>
</table>

8.11 С помощью контекстного меню по шаблону выделить все клоны



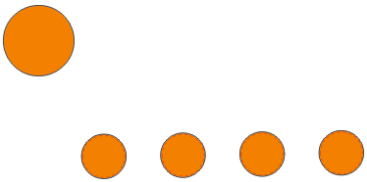
8.12 Применить команды Текст ► Простой текст ► Текст в рамку.

Получим аккуратно вписанный текст в круги-контейнеры



8.13 Уменьшить размер всех клонов. Для этого с помощью контекстного меню выделить клоны и уменьшить их.

Видно, что вписанный текст исчез.



8.14 Применить к выделенным клонам команды Текст ► Простой текст ► Текст в рамку

9 Разместите полученные вершины графа на карте (рисунок 25).

Рисунок 25 – Размещение клонов на карте

Для ускорения работы, точности размещения и быстрого перемещения по карте удобно использовать инструмент **Масштаб**  и навигатор, который находится в правом нижнем углу рабочего стола (рисунок 26).

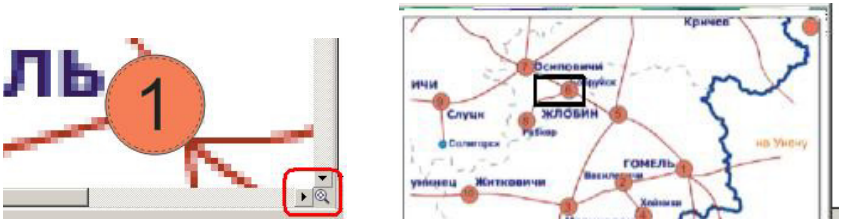


Рисунок 26 – Работа с инструментом Масштаб

10 Заготовьте шаблон восьмиугольный контейнер для текста. Добавьте восьмиугольнику контрастную заливку для лучшей видимости на фоне карты.

11 Создайте клоны контейнера для текста и разместите их на карте между вершинами графа (рисунок 27).

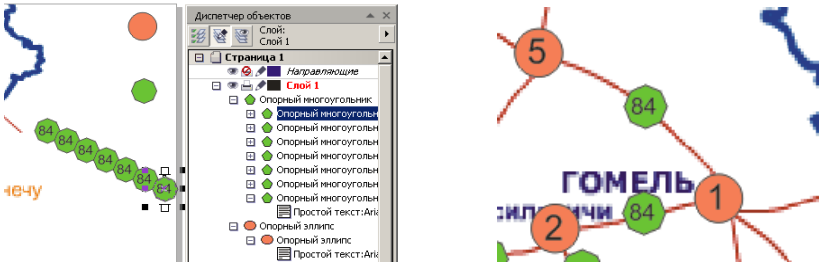


Рисунок 27 – Работа с шаблоном восьмиугольный контейнер для текста

12 Соедините узлы на контурах элементов инструментом **Соединительная линия** . Если карта мешает работать, можно включить режим **Вид►Каркас** (рисунок 28)**.** Отменить этот режим можно командой **Вид►Расширенный**.

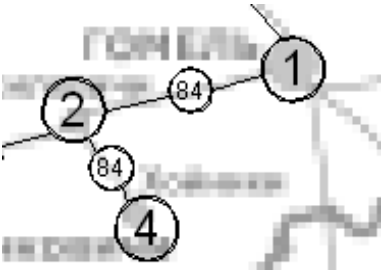


Рисунок 28 – Каркас исходного изображения с картой дорог

13 Отредактируйте веса рёбер. Отмените заливку и обводку шаблона контейнера для веса рёбер (рисунок 29). Подберите подходящую заливку шаблона вершины графа.

Рисунок 29 – Редактирование весов рёбер

14 Если разблокировать и удалить карту, получим изображение графа, аналогичное представленному фрагменту (рисунок 30).

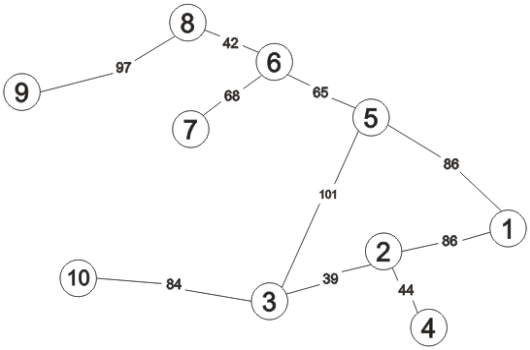


Рисунок 30 – Изображение графа

Контрольные вопросы

1 Создание и редактирование объектов в векторном редакторе.

2 Копирование, дублирование и клонирование объектов в векторном графическом редакторе.

3 Выравнивание и распределение векторных объектов.

4 Соединение узлов на контурах элементов.

3 БЫСТРОЕ СОЗДАНИЕ ИЗОБРАЖЕНИЙ ГРАФА С ПОМОЩЬЮ ПРОГРАММЫ ВЕКТОРНОЙ ГРАФИКИ MS VISIO

Редактор MS Visio является самой удобной программой для быстрого создания векторных изображений с помощью перемещения мышью готовых фигур в документ. Потом их можно масштабировать, деформировать, перетаскивать, размножать. Особенно удобно работать с ломанными или ступенчатыми стрелками и линиями связи – они имеют по несколько узловых точек, с помощью которых мышкой можно их изгибать и растягивать.

Основными объектами редактора являются: *документ, шаблон, трафарет, фигура* и *мастер*. Дадим краткое описание этих объектов.

1 *Документ* (drawing file): файл типа **.vsd**, в котором сохраняются *рисунки:* схемы, диаграммы и другие созданные пользователем изображения. Состоит из одного или нескольких листов.

2 *Шаблон* (template): файл типа **.vst**, является заготовкой документа и по структуре с ним одинаков. Все шаблоны, поставляемые с редактором Visio, разбиты на несколько категорий по принадлежности к определенной предметной области. Можно использовать встроенные шаблоны, создавать собственные или искать подходящие шаблоны на веб-сайте Office.com.

3 *Трафарет* (stencil): файл типа **.vss**, содержит набор фигур и мастеров определенного назначения (рисунок 31). Все трафареты, устанавливаемые при запуске Visio, находятся в папке Solutions и разделены на категории.

4 *Фигура* (shape): графический объект, созданный с помощью инструментов Visio, в частности – объект, полученный при перемещении мастера из трафарета на рабочий лист.

5 *Мастер* (master): в Visio так называется фигура на трафарете. Отметим, что в некоторых настройках Visio используются мастера в общепринятом смысле, например, мастер изменения цветовой схемы – Color Scheme.

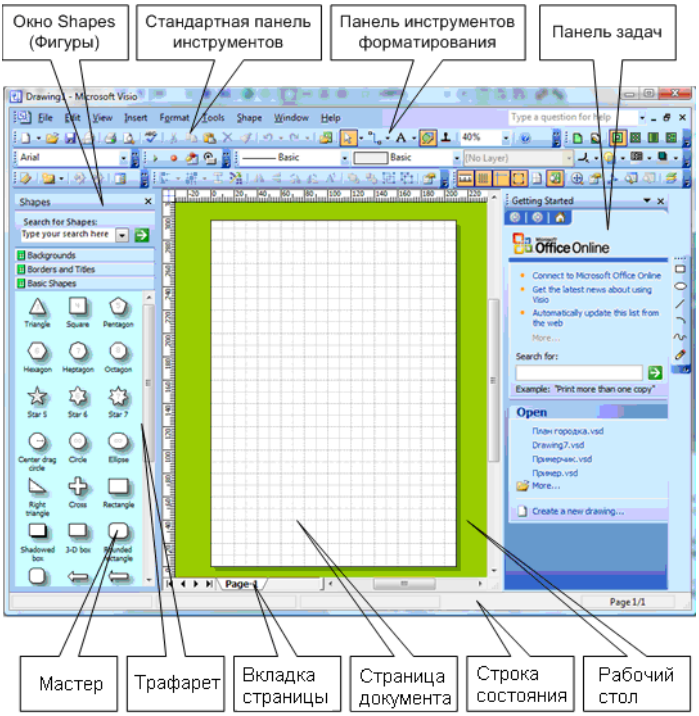
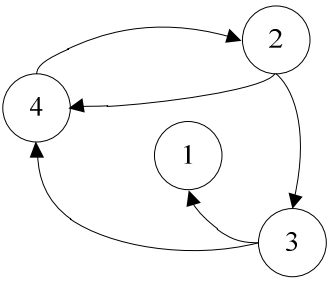


Рисунок 31 – Основные части окна рисования Visio

Задача. С помощью редактора MS Visio построить изображение графа:



Решение.

1 Откройте приложение MS Visio. Интерфейс редактора приближен к интерфейсу компонентов MS Office. Верхняя панель программы во многом повторяет панель Word и других компонентов Office.

2 Создайте новый файл. На вкладке **Файл** выберите **Создать**.

3 Сохраните файл. В дальнейшем не забывайте периодически сохранять сделанную работу для предотвращения потери данных из-за неустойчивой работы приложения или оборудования.

4 Для дальнейшей работы следует использовать окно **Фигуры** (Shapes), на трафарете которого расположены мастера (фигуры на трафарете) из различных электронных библиотек. Сначала выбираем нужную электронную библиотеку. Затем находим нужный мастер, нажимаем на него мышкой и «вытягиваем» на лист. Таким образом, получаем фигуру на странице документа.

Далее можно изменить размеры фигуры, цвет, добавить тень, написать текст.

Для соединения фигур используются соединительные линии, которые могут принимать любую форму. Эти линии при соприкосновении с узлами фигур фиксируются. При переносе фигуры соединительные линии будут сами следовать вслед за ней.

Контрольные вопросы

1 Программное обеспечение для создания и обработки векторных графических изображений.

2 Основные объекты редактора MS Visio.

4 ОБМЕН ДАННЫМИ МЕЖДУ ПРИЛОЖЕНИЯМИ ОС WINDOWS

Для выполнения учебных заданий удобно использовать возможности электронных таблиц. Затем данными из таблицы можно пользоваться в других приложениях. Рассмотрим несколько способов передачи данных в различные приложения.

1 С использованием буфера обмена.

Чтобы передать данные в другое приложение, нужно:

– выделить соответствующие ячейки;

– скопировать данные в буфер, используя для этого команду меню **Правка**-**Копировать**, либо комбинацию клавиш Ctrl+C.

Теперь в другом приложении можно «вынуть» данные из буфера с помощью команды меню **Правка**-**Вставить** или комбинации клавиш Ctrl+V.

2 Экспорт данных из табличного файла в формат CSV.

Текстовый формат CSV предназначен для представления табличных данных, где каждая строка файла – это одна строка таблицы. Значения отдельных колонок разграничиваются разделительным символом.

Порядок экспорта в формат CSV следующий:

– открыть документ Excel;

– выбрать команду меню **Файл-Сохранить как**;

– выбрать в окне тип файла CSV (разделители запятые) (*.csv);

– сохранить;

– при возникновении вопроса о подтверждении нажимаем **ДА** или **ОК**.

С файлом формата CSV могут работать приложения Excel, Блокнот и другие текстовые редакторы.

По умолчанию файл CSV открывается приложением Excel.

Для работы с данными в Блокноте можно в TotalCommander нажать клавишу **F4** или применить команду контекстного меню **Открыть с помощью**.

Контрольные вопросы

1 Способы передачи данных в различные приложения.

2 Экспорт данных из табличного файла в формат CSV.

3 Особенности открытия файла CSV.

5 ТЕОРИЯ ГРАФОВ И СТРУКТУРЫ ДАННЫХ В ПОИСКЕ КРАТЧАЙШИХ РАССТОЯНИЙ И ПУТЕЙ: ТЕОРЕТИЧЕСКИЙ МИНИМУМ

5.1 Элементы теории графов: невзвешенные графы

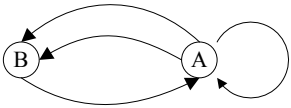
Граф можно представить как множество точек, некоторые пары которых соединены линиями.

Определение (строго). **Неориентированным графом** называется пара $G = (V, E)$ из конечного множества вершин V и множества ребер E, элементами которого являются пары вершин графа G:

Ориентированным графом или **орграфом** называется граф, на ребрах которого поставлены стрелки. Его ребра называются дугами.

Определение (строго). **Ориентированный граф** – это пара $G = (V, E)$ из конечного множества вершин V и множества дуг E, элементами которого являются упорядоченные пары вершин графа G:

В орграфе разрешаются: петли – дуги из вершины в себя саму; несколько дуг из одной вершины в другую (кратные дуги); «встречные» дуги (например, из *A* в *B* и из *B* в *A*):



Путем в графе называется конечная последовательность вершин (не обязательно различных), в которой всякие две соседние вершины соединены ребром.

Циклом называется путь, в котором первая и последняя вершины совпадают.

Длина пути – число ребер, по которым проходит этот путь.

В данном пособии для достижения понимания основных идей и быстрого старта в программирование не будем вводить дополнительную терминологию и различение таких понятий как «ребро – дуга», «путь – маршрут», «цикл –контур» и др.

Пример 1. Дан граф G1(рисунок 32).

Найти путь и его длину из вершины 1 в вершину 4.

Найти циклы из вершин 1 и 3.

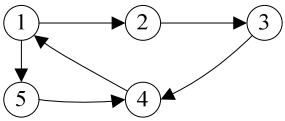


Рисунок 32 – Граф G1

Решение.

Путь из вершины 1 в вершину 4	Длина пути
1 – 2 – 3 – 4	3
1 – 5 – 4	2

Примеры циклов
1 – 2 – 3 – 4 – 1
1 – 5 – 4 – 1
3 – 4 – 1 – 2 – 3

Пример 2. Студент Иванов живет в городе Гомеле. Он выезжал на экскурсии в города Минск и Гродно.

Написать программу для подсчета количества знакомых ему городов.

Решение.

Представим данные в виде ориентированного графа G (рисунок 33).

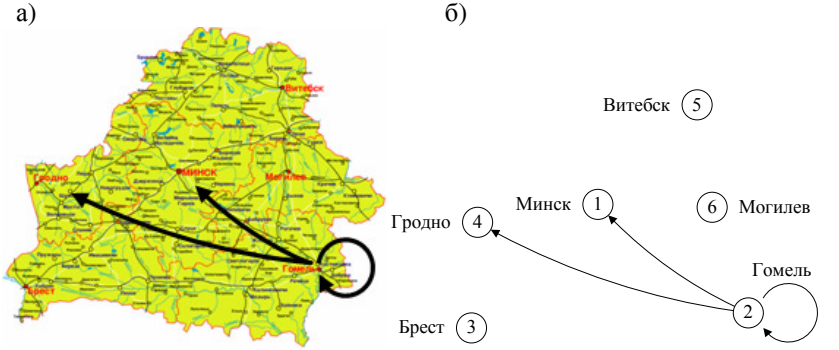


Рисунок 33 – Ориентированный граф G:
а – карта Республики Беларусь с наложением граф-схемы; *б* – изображение графа

Вершины графа можно именовать или нумеровать. Но для программирования вершины удобнее обозначать номерами.

Занесем в таблицу или матрицу смежности данные о знакомых Иванову городах: 0 – не был в городе, 1 – был в городе. Причем город Гомель студенту Иванову также знаком. Номер строки указывает, из какого города студент выезжал, а номер столбца – в какой город въезжал (рисунок 34).

Рисунок 34 – Матрица смежности для графа G

	1	2	3	4	5	6
1	0	0	0	0	0	0
2	1	1	0	1	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

Решение задачи сводится к нахождению суммы элементов во второй строке.

```
Program Summa;
 // В графе n вершин:
 const
   n=6;

 // Исходный граф:
 const G : array [1..n, 1..n] of byte =
   ((0, 0, 0, 0, 0, 0),
    (1, 1, 0, 1, 0, 0),
    (0, 0, 0, 0, 0, 0),
    (0, 0, 0, 0, 0, 0),
    (0, 0, 0, 0, 0, 0),
    (0, 0, 0, 0, 0, 0));
 // Вспомогательные переменные:
```

```
  var
    i, j, sum : integer;

BEGIN
  // Выводим матрицу смежности исходного графа по строчкам:
  writeln('Матрица смежности:');
  for i:=1 to n do begin
    for j:=1 to n do
      write(G[i, j],' ');
    writeln;
  end;
  // Суммируем элементы второй строки:
  sum:= 0;
  for j:=1 to n do
    sum:= sum + G[2, j];
  // Вывод количества знакомых Иванову городов:
  writeln('Количество знакомых городов = ', sum);
END.
```

Тест 1.

```
Матрица смежности:
0 0 0 0 0 0
1 1 0 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
Количество знакомых городов = 3
```

5.2 Обход ориентированного невзвешенного графа в ширину и глубину: основные понятия и различения

Граф называется **связным**, если для любых двух его вершин существует путь, начинающийся в первой из них и заканчивающийся во второй.

Связный граф называется **деревом**, если в нем не существует циклов.

Приведем примеры обхода орграфа типа «дерево» в глубину (рисунок 35) и ширину (рисунок 36).

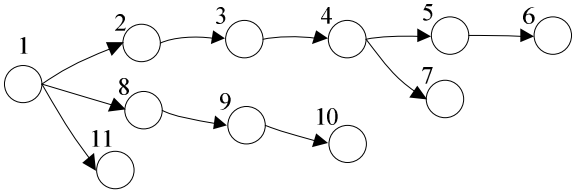


Рисунок 35 – Обход орграфа в глубину

Правило обхода орграфа в глубину удобно описывать с помощью структуры данных «стек», а в ширину – с помощью структуры данных «очередь».

Номера рядом с вершинами показывают порядок обхода вершин.

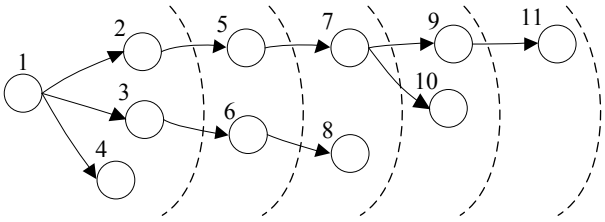


Рисунок 36 – Обход орграфа в ширину

Обход орграфа в ширину называют также **волновым алгоритмом**. Этот алгоритм позволяет быстро найти путь и его длину из указанной вершины до искомой. Его также используют при автоматизированной трассировке (разводке) печатных плат.

Контрольные вопросы

1 Сформулируйте понятие графа и приведите примеры графов.
2 Сформулируйте понятие ориентированного графа и приведите примеры.
3 Сформулируйте основные понятия теории графов.
4 Назовите матрицу, с помощью которой задается структура графа.

6 ПОИСК ПУТИ. ОБХОД НЕВЗВЕШЕННОГО ГРАФА В ГЛУБИНУ С ПОМОЩЬЮ СТРУКТУРЫ ДАННЫХ «СТЕК»

6.1 Теоретический минимум

Рассмотрим следующие понятия.

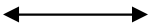
1 Структура данных стек.

2 Обход невзвешенного графа в глубину с помощью структуры данных «стек».

3 Поиск расстояний и путей из стартовой вершины в конечную вершину.

4 Поиск достижимых вершин из стартовой вершины.

Стек (**stack** – стопка) – это структура данных с последовательным доступом к элементам по принципу LIFO (Last In First Out) – последним пришел, первым ушел. То есть элементы из стека можно доставать только в порядке, обратном порядку добавления их в стек. Например, железнодорожный тупик: первым из тупика можно выгнать вагон, который был загнан туда последним:



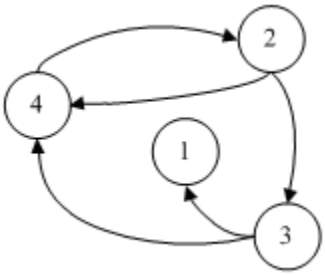
Обход графа в глубину сильно зависит от порядка, в котором просматриваются вершины соседние с текущей. В алгоритме обычно задают **правило обхода**, которое определяет, какую из соседних вершин следует выбрать на текущем шаге. Пропишем два правила:

1) соседей вершины перебирать в порядке **от меньшего номера к большему**;

2) соседей вершины перебирать в порядке **от большего номера к меньшему**.

Задание 1. а)

ДАНО. Невзвешенный орграф и пара вершин – стартовая и конечная.



НАДО. Используя алгоритм поиска в глубину, найти путь и его длину из стартовой вершины в конечную вершину или установить, что пути нет. Выпишите последовательно все изменения стека за время работы алгоритма.

Выполните задание с учетом правила обхода 1, затем с учетом правила обхода 2.

Решение.

Пусть стартовая вершина = 2, а конечная вершина = 4.

Запишем матрицу смежности для указанного графа:

		Куда			
		1	2	3	**4**
Откуда	1	0	0	0	0
	2	0	0	1	1
	3	1	0	0	1
	4	0	1	0	0

Поиск пути. Правило 1.

Поместили в стек вершину 2. Отметили ее как посещенную:

2 ←

У вершины 2 два соседа: 3 и 4. Выбрали вершину с меньшим номером 3 и поместили ее в стек. Отметили вершину 3 как посещенную:

2 3 ←

У вершины 3 два соседа: 1 и 4. Выбрали вершину с меньшим номером 1. Отметили вершину 1 как посещенную:

2 3 1 ←

У вершины 1 нет соседей. Удалили ее из стека. Вернулись в вершину 3:

2 3 →

У вершины 3 один непосещенный сосед – 4. Выбрали эту вершину. Отметили вершину 4 как посещенную:

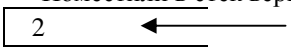
2 3 4 ←

Выписываем ответ из стека.

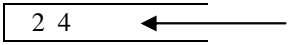
Ответ. Путь из вершины 2 в вершину 4 будет 2-3-4. Длина пути: 2.

Поиск пути. Правило 2.

Поместили в стек вершину 2. Отметили ее как посещенную:



У вершины 2 два соседа: 3 и 4. Выбрали вершину с большим номером 4 и поместили ее в стек. Отметили вершину 4 как посещенную:



Выписываем ответ из стека.

Ответ. Путь из вершины 2 в вершину 4 будет 2-4. Длина пути: 1.

Из ответов видно, что порядок обхода влияет на найденные пути.

Задание 1. б)

ДАНО. Невзвешенный орграф и стартовая вершина.

НАДО. Найти все вершины, достижимые из стартовой.

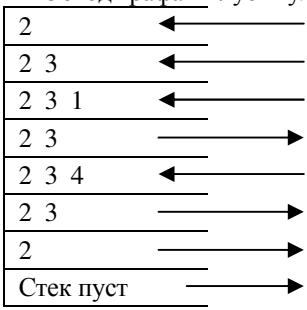
Записать, какие вершины орграфа и в каком порядке попадали в стек, выписать последовательно все изменения стека за время работы алгоритма. Для каждой вершины, попавшей в стек, указать ее предка.

Решение.

Обход невзвешенного графа в глубину с помощью структуры данных «стек». Правило 1

Пусть стартовая вершина = 2.

Обход графа в глубину. Вывод состояния стека на каждом шаге:



Укажем предков для каждой вершины:

для вершины 1 – предок = 3;

для вершины 2 – предок = 0 (указывает на отсутствие предка);

для вершины 3 – предок = 2;

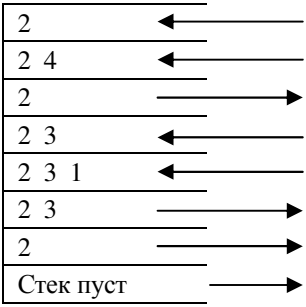
для вершины 4 – предок = 3.

Порядок попадания вершин в стек >> 2 3 1 4.

Ответ. Список достижимых вершин {3, 1, 4}. Все вершины достижимы.

Обход невзвешенного графа в глубину с помощью структуры данных «стек». Правило 2

Обход графа в глубину. Вывод состояния стека на каждом шаге:



Укажем предков для каждой вершины:

для вершины 1 – предок = 3;

для вершины 2 – предок = 0 (указывает на отсутствие предка);

для вершины 3 – предок = 2;

для вершины 4 – предок = 2.

Порядок попадания вершин в стек >> 2 4 3 1.

Ответ. Список достижимых вершин {4, 3, 1}. Все вершины достижимы.

6.2 Программирование задания по правилу 1

Решение задания 1.а) и 1.б) находится в одной программе, но в разных процедурах.

В процедуре **poisk_puti** находится решение задачи 1.а).

В процедуре **obhod** находится решение задачи 1.б).

Программа выполняет обход в глубину из стартовой вершины и:

– выводит состояние стека на каждом шаге;

– находит путь и его длину из стартовой вершины в конечную;

– перечисляет предков для каждой вершины, попавшей в стек;

– выводит порядок попадания вершин в стек, причем это и есть список достижимых вершин из стартовой вершины.

Программа 1.

```
Program DepthFirstSearch;
  // В графе n вершин:
  const   n=4;
  // Исходный граф:
  const graph: array[1..n,1..n] of byte =
  ((0, 0, 0, 0),
   (0, 0, 1, 1),
   (1, 0, 0, 1),
   (0, 1, 0, 0));
 var
  // Вспомогательные переменные:
  i, j, r, start, finish, p : integer;
```

```
  // Массив для пометок посещенных вершин:
  visited: array[1..n] of boolean;
  // Массив для стека и указатель на вершину стека:
  stek : array [1..n] of byte;
  uk   : byte;
  // Массив предков для каждой вершины, попавшей в стек:
  predok :  array [1..n] of integer;
  // Порядок попадания вершин в стек в ходе поиска:
  dfs : array [1..n] of integer;
  // Длина пути из стартовой вершины в конечную:
  dlina : integer;

Procedure poisk_puti;    // Решение задачи 1.а)
begin
 // Инициализация стека:
 for i:=1 to n do stek[i]:= 0;
 // Помечаем вершины графа как непосещенные:
 for i:=1 to n do visited[i]:=false;
 {Пока стек не пуст и не найдена конечная вершина,
 удаляем или добавляем вершину. При этом указатель на вершину стека
 уменьшается или увеличивается.}
 uk:=1;
 stek[uk]:=start; // Порядок попадания вершин в стек – вершина start.
 visited[start]:=true;

 while (uk<>0) and (stek[uk]<>finish) do begin
  {Просматриваем столбцы строки от меньшего номера к большему.
  Если в просматриваемой строке столбец равен 0 или вершина уже
  посещена, в то же время не найдена конечная вершина, то
  переходим к следующему номеру столбца.}
  r:=1;
  while (r<=n) and (start<>finish)
  and ( (graph[stek[uk],r]=0) or (visited[r]) )
  do begin r:=r+1;
  end;

  {Если найдена непосещещенная вершина, увеличиваем указатель
  верхушки стека и помещаем вершину в стек.
  Иначе указатель верхушки стека уменьшаем.}
  if (r<=n) then begin
    uk:=uk+1;
    stek[uk]:=r;
    visited[r]:=true; // Помечаем вершины графа как посещенные.
  end
  else begin uk:=uk-1; end;
 end;
```

```
  if uk=0 then writeln ('Пути нет')
  else begin
    for i:=1 to uk do write(stek[i]:3);
    writeln('  Длина пути = ', uk-1);
  end;
 end;
```

Procedure obhod; *// Решение задачи 1.б)*

```
begin
  // Инициализация стека:
  for i:=1 to n do  stek[i]:= 0;
  // Помечаем вершины графа как непосещенные:
  for i:=1 to n do   visited[i]:=false;
  // Заполняем  массив предков нулями, т.е. 0 – предков нет:
  for i:=1 to n do  predok [i]:=0;

  { Пока стек не пуст, удаляем или добавляем вершину.
  При этом указатель на вершину стека уменьшается или увеличивается.}
  uk:=1;
  stek[uk]:=start;
  visited[start]:=true;
  p:=1;          // Количество попавших вершин в стек – 1.
  dfs[p]:=start;  // Порядок попадания вершин в стек – вершина start.
  while uk<>0 do begin
    for i:=1 to uk do write(stek[i]:3);// Вывод состояния стека.
    writeln;

    { Просматриваем столбцы строки в порядке от меньшего номера
    к большему.  Если в просматриваемой строке столбец равен 0 или
    вершина уже посещена, то переходим к следующему номеру столбца.}
    r:=1;
    while (r<=n) and ((graph[stek[uk],r]=0) or (visited[r]))
      do begin r:=r+1;
    end;

    { Если найдена непосещеченная вершина, увеличиваем указатель
    верхушки стека  и помещаем вершину в стек.
    Иначе указатель верхушки стека уменьшаем. }
    if (r<=n)then begin
      uk:=uk+1;
      stek[uk]:=r;
      visited[r]:=true; // Помечаем вершины графа как посещенные.
      // Указываем предков для каждой вершины, попавшей в стек:
      predok [r]:=stek[uk-1];
      p:=p+1;          // Количество попавших вершин в стек.
```

```
      dfs[p]:=r;       // Порядок попадания вершин в стек.
    end
    else begin uk:=uk-1;
    end;
  end;
end;

BEGIN
  // Выводим матрицу смежности исходного графа по строчкам:
  writeln('Матрица смежности:');
  for i:=1 to n do begin
    for j:=1 to n do
      write(graph[i, j],' ');
    writeln;
  end;

  write('Стартовая вершина = '); read(start);
  write('Конечная вершина  = '); read(finish);

  writeln('Обход графа в глубину. Вывод состояния стека на каждом шаге. ');
  writeln('Перебор соседей вершины от меньшего номера к большему.');
  //Вызов процедуры обхода графа для поиска достижимых вершин:
  obhod;

  writeln('Вывод предков для каждой вершины ');
  for i:=1 to n do writeln ('Для вершины-', i:3, '-предок=', predok [i]:3);

  write('Порядок попадания вершин в стек >>');
  for i:=1 to p do write(dfs[i]:3);
  writeln;

  writeln( 'Вывод пути из вершины-', start, ' в вершину-',finish);
  // Вызов процедуры обхода графа в глубину для поиска пути:
  poisk_puti;

END.
```

6.3 Тестирование программы по правилу 1

Тест 1.

```
Матрица смежности:
0 0 0 0
0 0 1 1
1 0 0 1
0 1 0 0
Стартовая вершина = 2
Конечная вершина  = 4
Обход графа в глубину. Вывод состояния стека на каждом шаге.
Перебор соседей вершины от меньшего номера к большему.
```

```
  2
  2  3
  2  3  1
  2  3
  2  3  4
  2  3
  2
Вывод предков для каждой вершины
Для вершины-  1-предок=  3
Для вершины-  2-предок=  0
Для вершины-  3-предок=  2
Для вершины-  4-предок=  3
Порядок попадания вершин в стек >>  2  3  1  4
Вывод пути из вершины-2 в вершину-4
  2  3  4  Длина пути = 2
```

Тест 2.

```
Матрица смежности:
0 0 0 0
0 0 1 1
1 0 0 1
0 1 0 0
Стартовая вершина = 1
Конечная вершина  = 2
Обход графа в глубину. Вывод состояния стека на каждом шаге.
Перебор соседей вершины от меньшего номера к большему.
  1
Вывод предков для каждой вершины
Для вершины-  1-предок=  0
Для вершины-  2-предок=  0
Для вершины-  3-предок=  0
Для вершины-  4-предок=  0
Порядок попадания вершин в стек >>  1
Вывод пути из вершины-1 в вершину-2
Пути нет
```

6.4 Программирование задания по правилу 2

В программе 2 находится решение задания 1.а) и 1.б). Причем перебор соседей вершины производится в порядке **от большего номера к меньшему.**

В процедуре **poisk_puti** находится решение задачи 1.а).

В процедуре **obhod** находится решение задачи 1.б).

Программа выполняет обход в глубину из стартовой вершины и:

– выводит состояние стека на каждом шаге;

– находит путь и его длину из стартовой вершины в конечную;

– перечисляет предков для каждой вершины, попавшей в стек;

– выводит порядок попадания вершин в стек, причем это и есть список достижимых вершин из стартовой вершины.

Программа 2.

```
Program DepthFirstSearch;
   // В графе n вершин:
   const    n=4;
  // Исходный граф:
   const graph: array[1..n,1..n] of byte =
   ((0, 0, 0, 0),
    (0, 0, 1, 1),
    (1, 0, 0, 1),
    (0, 1, 0, 0));
  var
   // Вспомогательные переменные:
   i, j, r, start, finish, p : integer;
   // Массив для пометок посещенных вершин:
   visited: array[1..n] of boolean;
   // Массив для стека и указатель на вершину стека:
   stek : array [1..n] of byte;
   uk   : byte;
   // Массив предков для каждой вершины, попавшей в стек:
   predok :  array [1..n] of integer;
   // Порядок попадания вершин в стек в ходе поиска:
   dfs : array [1..n] of integer;
   // Длина пути из стартовой вершины в конечную:
   dlina : integer;

Procedure poisk_puti;    // Решение задачи 1.а)
begin
  // Инициализация стека:
  for i:=1 to n do stek[i]:= 0;
  // Помечаем вершины графа как непосещенные:
  for i:=1 to n do visited[i]:=false;
  {Пока стек не пуст и не найдена конечная вершина,
  удаляем или добавляем вершину. При этом указатель на вершину стека
  уменьшается или увеличивается.}
  uk:=1;
  stek[uk]:=start; // Порядок попадания вершин в стек – вершина start.
  visited[start]:=true;

  while (uk<>0) and (stek[uk]<>finish) do begin
    {Просматриваем столбцы строки от большего номера к меньшему.
    Если в просматриваемой строке столбец равен 0 или вершина уже
    посещена, в то же время не найдена конечная вершина, то
    переходим к следующему номеру столбца.}
    r:=n;
    while (r>=1) and (start<>finish)
    and ( (graph[stek[uk],r]=0) or (visited[r]) )
```

```
  do begin r:=r-1;
  end;

  { Если найдена непосещещенная вершина, увеличиваем указатель
  верхушки стека и помещаем вершину в стек.
  Иначе указатель верхушки стека уменьшаем.}
  if (r>0) then begin
    uk:=uk+1;
    stek[uk]:=r;
    visited[r]:=true; // Помечаем вершины графа как посещенные.
  end
  else begin uk:=uk-1; end;
 end;

 if uk=0 then writeln ('Пути нет')
 else begin
  for i:=1 to uk do write(stek[i]:3);
  writeln('  Длина пути = ', uk-1);
 end;
end;

Procedure obhod;      // Решение задачи 1.б)
begin
 // Инициализация стека:
 for i:=1 to n do  stek[i]:= 0;
 // Помечаем вершины графа как непосещенные:
 for i:=1 to n do   visited[i]:=false;
 // Заполняем  массив предков нулями, т. е. 0 – предков нет:
 for i:=1 to n do  predok [i]:=0;

 { Пока стек не пуст, удаляем или добавляем вершину.
 При этом указатель на вершину стека уменьшается или увеличивается.}
 uk:=1;
 stek[uk]:=start;
 visited[start]:=true;
 p:=1;          // Количество попавших вершин в стек – 1.
 dfs[p]:=start;  // Порядок попадания вершин в стек – вершина start.
 while uk<>0 do begin
  for i:=1 to uk do write(stek[i]:3);// Вывод состояния стека.
  writeln;

  { Просматриваем столбцы строки в порядке от большего номера
  к меньшему.  Если в просматриваемой строке столбец равен 0 или
  вершина уже посещена, то переходим к следующему номеру столбца.}
  r:=n;
  while (r>=1) and ((graph[stek[uk],r]=0) or (visited[r]))
   do begin r:=r-1;
```

```
    end;

    { Если найдена непосещещенная вершина, увеличиваем указатель
    верхушки стека  и помещаем вершину в стек.
    Иначе указатель верхушки стека уменьшаем.}
    if (r>0)then begin
      uk:=uk+1;
      stek[uk]:=r;
      visited[r]:=true; // Помечаем вершины графа как посещенные.
      // Указываем предков для каждой вершины, попавшей в стек:
      predok [r]:=stek[uk-1];
      p:=p+1;          // Количество попавших вершин в стек.
      dfs[p]:=r;       // Порядок попадания вершин в стек.
    end
    else begin uk:=uk-1;
    end;
  end;
end;

BEGIN
  // Выводим матрицу смежности исходного графа по строчкам:
  writeln('Матрица смежности:');
  for i:=1 to n do begin
    for j:=1 to n do
      write(graph[i, j],' ');
    writeln;
  end;

  write('Стартовая вершина = '); read(start);
  write('Конечная вершина  = '); read(finish);

  writeln('Обход графа в глубину. Вывод состояния стека на каждом шаге. ');
  writeln('Перебор соседей вершины от большего номера к меньшему. ');
  // Вызов процедуры обхода графа для поиска достижимых вершин:
  obhod;

  writeln('Вывод предков для каждой вершины ');
  for i:=1 to n do writeln ('Для вершины-', i:3, '-предок=', predok [i]:3);

  write('Порядок попадания вершин в стек >>');
  for i:=1 to p do write(dfs[i]:3);
  writeln;

  writeln( 'Вывод пути из вершины-', start, ' в вершину-',finish);
  // Вызов процедуры обхода графа в глубину для поиска пути:
  poisk_puti;

END.
```

6.5 Тестирование программы по правилу 2

Тест 1

```
Матрица смежности:
0 0 0 0
0 0 1 1
1 0 0 1
0 1 0 0
Стартовая вершина = 2
Конечная вершина  = 4
Обход графа в глубину. Вывод состояния стека на каждом шаге.
Перебор соседей вершины от большего номера к меньшему.
 2
 2  4
 2
 2  3
 2  3  1
 2  3
 2
Вывод предков для каждой вершины
Для вершины-  1-предок=  3
Для вершины-  2-предок=  0
Для вершины-  3-предок=  2
Для вершины-  4-предок=  2
Порядок попадания вершин в стек >>  2  4  3  1
Вывод пути из вершины-2 в вершину-4
4  Длина пути = 1
```

Тест 2

```
Матрица смежности:
0 0 0 0
0 0 1 1
1 0 0 1
0 1 0 0
Стартовая вершина = 3
Конечная вершина  = 2
Обход графа в глубину. Вывод состояния стека на каждом шаге.
Перебор соседей вершины от большего номера к меньшему.
 3
 3  4
 3  4  2
 3  4
 3
 3  1
 3
Вывод предков для каждой вершины
Для вершины-  1-предок=  3
Для вершины-  2-предок=  4
Для вершины-  3-предок=  0
```

```
Для вершины- 4-предок= 3
Порядок попадания вершин в стек >> 3 4 2 1
Вывод пути из вершины-3 в вершину-2
 3 4 2 Длина пути = 2
```

Контрольные вопросы

1 Сформулируйте понятие структуры данных «стек».

2 Приведите примеры организации чего-либо по принципу стека.

3 Сформулируйте понятие обхода невзвешенного графа в глубину с помощью структуры данных «стек».

4 Влияет ли правило обхода на получение результата?

7 ПОИСК КРАТЧАЙШЕГО ПУТИ. ОБХОД НЕВЗВЕШЕННОГО ГРАФА В ШИРИНУ С ПОМОЩЬЮ СТРУКТУРЫ ДАННЫХ «ОЧЕРЕДЬ»

7.1 Теоретический минимум

Обход графа в ширину – один из основных алгоритмов на графах, который находит путь кратчайшей длины в невзвешенном графе. Этот путь содержит наименьшее число ребер. С помощью этого обхода можно также найти все достижимые вершины из стартовой вершины.

Рассмотрим описание алгоритма на следующей задаче.

Задание 1.

ДАНО. Невзвешенный граф и номер стартовой вершины. Граф может быть как ориентированным, так и неориентированным.

НАДО. Найти кратчайший путь и его длину от стартовой вершины до конечной вершины с помощью обхода графа в ширину.

Решение.

Пусть стартовая вершина = 2, а конечная вершина = 1.

Запишем матрицу смежности для графа:

		Куда			
		1	2	3	4
	1	0	0	0	0
Откуда	**2**	0	0	1	1
	3	1	0	0	1
	4	0	1	0	0

На нулевом шаге «поджигаем» только одну вершину.

Поместили в очередь вершину 2. Отметили ее как посещенную. У стартовой вершины нет предка и длина пути пока равна 0:

Предок	0
Потомок	2
Длина пути	0

На каждом следующем шаге огонь с каждой уже горящей вершины перекидывается на всех ее соседей. То есть каждый следующий шаг расширяет «кольцо огня» на единицу (см. рисунок 36).

Алгоритм представляет собой цикл: пока очередь не пуста или не достигнута конечная вершина, достать из начала очереди одну вершину, просмотреть все ребра, исходящие из этой вершины, и если какие-то из соседних вершин еще не «горят», то «поджечь» их и поместить в конец очереди.

Обход завершается, когда в очередь попадет конечная вершина или очередь станет пустой, а конечная вершина не нашлась. Второе означает, что пути в эту вершину нет.

В каждой задаче уточняется, кто является «соседом». В нашем задании соседями будут считаться ненулевые элементы текущей строки из матрицы смежности.

В начале очереди находится вершина 2, и она станет предком. Просматриваем строку 2 в матрице смежности и записываем номера столбцов как потомков в конец очереди. Причем записываем только непосещенные вершины. Длина пути на этом шаге увеличивается на 1:

Предок	0	2	2
Потомок	2	3	4
Длина пути	0	1	1

Далее в начале очереди находится вершина 3, и она станет предком. Просматриваем строку 3 в матрице смежности и записываем номера столбцов как потомков в конец очереди. Причем записываем только непосещенные вершины. Длина пути на этом шаге увеличивается еще на 1:

Предок	0	2	2	3
Потомок	2	3	4	1
Длина пути	0	1	1	2

В очередь попала конечная вершина, следовательно, обход завершен. Причем до вершины дошли кратчайшим путем.

Путь выписываем с конца очереди: от потомка к предку до достижения предка с номером 0.

Предок	0	2	2	3
Потомок	2	3	4	1
Длина пути	0	1	1	2

Ответ. Кратчайший путь от вершины 2 до вершины 1: 2-3-1

Длина пути равна 2.

7.2 Программирование

С помощью трех одномерных массивов создаем очередь, в которую будем помещать горящие вершины.

В массиве predok будем хранить начало ребра.

В массиве potomok будем хранить конец ребра.

В массиве длин кратчайших путей dlina хранится длина пути из стартовой вершины до текущего потомка.

В массиве visit будем отмечать посещенные вершины.

Изначально в очередь помещается только стартовая вершина s. Помечаем ее как посещенную Visit[s] = true. Для всех остальных вершин Visit[i] = false.

Затем алгоритм проходит описанный ранее цикл поиска кратчайшего пути с помощью структуры данных «очередь».

В очереди компактно хранится информация для восстановления кратчайшего пути. Длину пути берем из массива dlina. Для восстановления кратчайшего пути достаточно пройти по массиву потомков potomok, в котором для каждой вершины хранится номер вершины предка predok, то есть из которой мы попали в эту вершину.

```
Program BreadthFirstearch;
 // В графе n вершин:
 const    n=4;
 // Исходный граф:
 const g: array[1..n,1..n] of byte =
   ((0, 0, 0, 0),
    (0, 0, 1, 1),
    (1, 0, 0, 1),
    (0, 1, 0, 0));
 Var
   // Массивы для  очереди
   predok    :  array [0..n*n] of integer;
   potomok :  array [0..n*n] of integer;
   dlina       :  array [0..n*n] of integer;

   // Вспомогательные переменные:
   i, j, st, qbegin, qend, start, finish, vihod : integer;
   // Массив для отметок посещенных вершин:
   visit  : array [1..n] of boolean;
   // Массив для хранения пути:
   path : array [1..n] of integer;

BEGIN
 // Выводим матрицу смежности исходного графа по строчкам:
 writeln('Матрица смежности:');
 for i:=1 to n do begin
   for j:=1 to n do
     write(g[i, j],' ');
   writeln;
 end;

 // Выводим  список ребер:
 writeln( 'Список ребер =' );
```

```
for i:=1 to n do begin
  for j:=1 to n do
    if g[i, j]<>0 then write ( '(',  i,  ',',  j,  ')',  ',' );
  writeln;
end;

for i:= 1 to n do  visit[i]:= false;              // Вершины не посещались.

write ( 'Стартовая вершина->' ); readln ( start );
write ( 'Конечная вершина-> ' ); readln ( finish );
qbegin:= 0; qend:= 1;  //Начало очереди = 0, конец очереди = 1.
predok[0]:= 0;
potomok[0]:= start;     // Кладем стартовую вершину в очередь.
visit[start]:= true;         // Отмечаем стартовую вершину как посещенную.
dlina[0]:= 0;
if start<>finish then vihod:=0 else vihod:=1;

// Пока конечная вершина не найдена, заполняем очередь:
while (vihod=0) and (qbegin<> qend)  do begin
  predok[qend]:= potomok[qbegin];

  {Из исходной матрицы G помещаем в очередь всех соседей,
  причем соседи – ненулевые элементы из строки с номером predok[qend]}
  st:= 1;
  while (st<=n) and (vihod=0) do begin
    if (G[predok[qend], st]= 1) and (not visit[st]) then begin
      potomok[qend]:=st;
      dlina[qend]:= dlina[qbegin]+1;
      visit[st]:= true;

      if (potomok[qend]<>finish)  then begin
        qend:= qend+1;                    // Увеличиваем конец очереди.
        predok[qend]:= potomok[qbegin];
      end
      else vihod:= 1;                     // Конечная вершина найдена.
    end;
    st:= st+1;
  end;
  qbegin:= qbegin+1;                      // Увеличиваем начало очереди.
end;

// Выводим состояние очереди от стартовой вершины до финишной:
Writeln ( 'Состояние очереди ');
for i:= 0 to qend do begin write ( predok[i], ' ' );   end;  writeln;
for i:= 0 to qend do begin write ( potomok[i], ' ' ); end;  writeln;
for i:= 0 to qend do begin write ( dlina[i], ' ' );       end;  writeln;
```

```
  // Выводим длину пути:
  writeln ( 'Длина пути ', dlina[qend] );

  // Восстанавливаем путь из очереди:
  if dlina[qend]=0  then writeln ( 'Пути нет, вершина ', finish, ' недостижима' )
  else begin                              // С конца очереди выписываем путь:
    i:= 1;    j:= qend;   path[i]:= potomok[j];
    while ( potomok[j]<>start ) do begin
      i:= i+1;
      path[i]:= predok[j];
      while path[i]<> potomok[j]  do
        j:=j-1;
    end;

    // Выводим путь:
    writeln ( 'Путь от вершины ', start, ' до вершины ', finish);
    for j:= i downto 1 do write ( path[j], ' '  ); writeln;
  end;
END.
```

7.3 Тестирование

Тест 1.

```
0 0 0 0
0 0 1 1
1 0 0 1
0 1 0 0
Список ребер =
(2,3),(2,4),
(3,1),(3,4),
(4,2),
Стартовая вершина->1
Конечная вершина->2
Состояние очереди
0 1
1 0
0 0
Длина пути 0
Пути нет, вершина 2 недостижима
```

Тест 2.

```
0 0 0 0
0 0 1 1
1 0 0 1
0 1 0 0
Список ребер =
(2,3),(2,4),
(3,1),(3,4),
(4,2),
Стартовая вершина->2
Конечная вершина->1
Состояние очереди
0 2 2 3
2 3 4 1
0 1 1 2
Длина пути 2
Путь от вершины 2 до вершины 1
2 3 1
```

Тест 3.

```
0 0 0 0
0 0 1 1
1 0 0 1
0 1 0 0
Список ребер =
(2,3),(2,4),
(3,1),(3,4),
(4,2),
Стартовая вершина->2
Конечная вершина->3
Состояние очереди
0 2
2 3
0 1
Длина пути 1
Путь от вершины 2 до вершины 3
2 3
```

Тест 4.

```
0 0 0 0
0 0 1 1
1 0 0 1
0 1 0 0
Список ребер =
(2,3),(2,4),
(3,1),(3,4),
(4,2),
Стартовая вершина->2
Конечная вершина->4
Состояние очереди
0 2 2
2 3 4
0 1 1
Длина пути 1
Путь от вершины 2 до вершины 4
2 4
```

Тест 5.

```
0 0 0 0
0 0 1 1
1 0 0 1
0 1 0 0
Список ребер =
(2,3),(2,4),
(3,1),(3,4),
(4,2),
Стартовая вершина->3
Конечная вершина->2
Состояние очереди
0 3 3 4
3 1 4 2
0 1 1 2
Длина пути 2
Путь от вершины 3 до вершины 2
3 4 2
```

Тест 6.

```
0 0 0 0
0 0 1 1
1 0 0 1
0 1 0 0
Список ребер =
(2,3),(2,4),
(3,1),(3,4),
(4,2),
Стартовая вершина->4
Конечная вершина->1
Состояние очереди
0 4 2 3
4 2 3 1
0 1 2 3
Длина пути 3
Путь от вершины 4 до вершины 1
4 2 3 1
```

Контрольные вопросы

1 Сформулируйте понятие структуры данных «очередь».

2 Приведите примеры организации чего-либо по принципу очереди.

3 Сформулируйте понятие обхода невзвешенного графа в ширину с помощью структуры данных «очередь».

8 ПОИСК КРАТЧАЙШИХ ПУТЕЙ НА ВЗВЕШЕННОМ ГРАФЕ ИЗ ОДНОЙ ВЕРШИНЫ ВО ВСЕ ОСТАЛЬНЫЕ С ПОМОЩЬЮ АЛГОРИТМА ДЕЙКСТРЫ

8.1 Элементы теории графов: взвешенные графы

Рассмотрим следующие понятия.

1 Взвешенный граф.

2 Матрица весов.

3 Матрица расстояний.

4 Кратчайший путь из одной вершины в другую во взвешенном графе.

Граф называется **взвешенным**, если каждому ребру соответствует какое-то число (вес). Взвешенным может быть как ориентированный, так и неориентированный граф. Только у неориентированного графа не может быть двух рёбер, соединяющих одни и те же вершины.

Матрицей весов называется квадратная матрица размерности n (где n – количество вершин), элемент которой, стоящий в i строке и j столбце, определяется по правилу

$$a_{ij} = \begin{cases} 0, \textit{ если вершины с номерами } i \textit{ и } j \textit{ совпадают}, \\ \infty, \textit{ если вершины с номерами } i \textit{ и } j \textit{ не смежны}, \\ \textit{вес ребра}, \textit{ если вершины с номерами } i \textit{ и } j \textit{ смежны}. \end{cases}$$

Знак ∞ обозначает, что нет ребра между двумя вершинами.

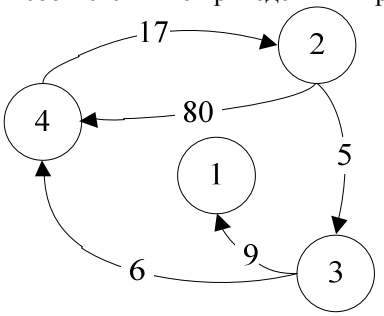
Далее матрицу весов используют как **матрицу расстояний** и записывают в нее кратчайшие расстояния между двумя вершинами, если есть путь, их связывающий.

Кратчайший путь из одной вершины в другую – это такой путь по рёбрам, где сумма весов пройденных рёбер будет минимальна.

Пример. Дан взвешенный граф G. Представить его как матрицу весов.

Решение.

В соответствии с приведенным правилом заполним матрицу весов.



	1	2	3	4
1	0	∞	∞	∞
2	∞	0	5	80
3	9	∞	0	6
4	∞	17	∞	0

8.2 Алгоритм Дейкстры

Задание 1.

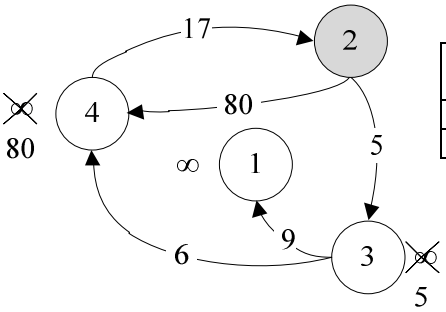
ДАНО. Взвешенный орграф и стартовая вершина. Все веса ребер неотрицательные числа, так как алгоритм Дейкстры с отрицательными весами не работает.

НАДО. Используя алгоритм Дейкстры, найти кратчайший путь и его длину из стартовой вершины до всех остальных.

Выписать последовательно все изменения в матрице расстояний за время работы алгоритма.

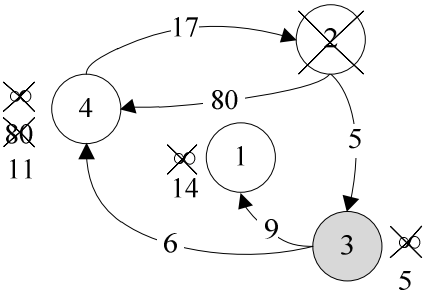
Решение.

Пусть стартовая вершина = 2. Следовательно, будем работать со строкой 2 из матрицы весов. Каждой вершине сопоставим метку. В самом начале это будут веса ребер или знак ∞, который говорит о том, что расстояния до других вершин пока неизвестны. Все вершины графа помечаются как непосещённые.



Текущая вершина	1	3	4	Очередь с приоритетом
из 2	∞	5	80	min(5,80) = 5
из 3				

Алгоритм работает пошагово: на каждом шаге он выделяет одну вершину с меньшей меткой и от нее пытается уменьшить метки у соседей. Затем вершина вычеркивается. Работа алгоритма завершается, когда все вершины вычеркнуты:

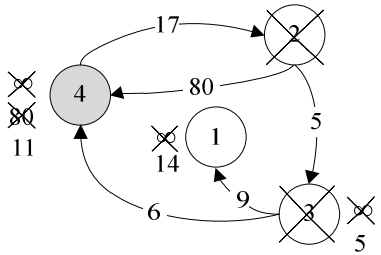


Текущая вершина	1	3	4	Очередь с приоритетом
из 2	∞	5	80	min(5,80) = 5
из 3	14		11	min(14,11) = 11
из 4				

На данном шаге просмотрели соседей из вершины 3. Из 2-й до 3-й доходили за 5. Далее можно пройти к 1-й за 5 + 9 = 14, а к 4-й за 5 + 6 = 11. У 1-й вершины сменили метку с ∞ на 14. У 4-й вершины сравнили предыдущую метку 80 и новое расстояние 11. Так как 80 > 11, меняем метку на меньшее значение. Нашли более короткий путь из 2-й до 4-й за 11 через вершину 3.

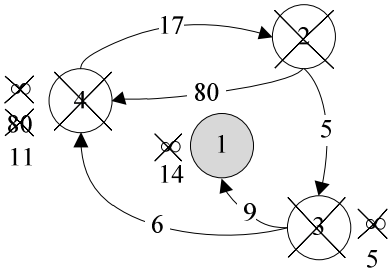
Чтобы выбрать следующую вершину, надо найти самую малую метку среди просмотренных, но не вычеркнутых вершин. Такая структура данных называется «**очередь с приоритетами**».

В очереди находятся вершина 1 с меткой 14 и вершина 4 с меткой 11. min (14,11) = 11. Следовательно, алгоритм будет работать из вершины 4:



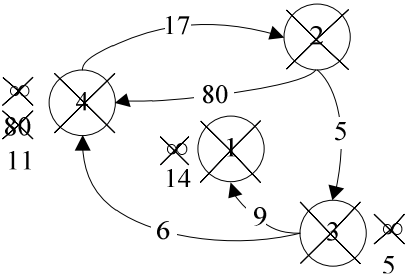
Текущая вершина	1	3	4	Очередь с приоритетом
из 2	∞	5	80	min(5,80) = 5
из 3	14		11	min(14,11) = 11
из 4				min(14) = 14
из 1				

На данном шаге просмотрели соседей из вершины 4. Так как в вычеркнутые вершины хода нет, то метки нигде изменить не удалось. Выбираем следующую вершину из очереди – 1:



Текущая вершина	1	3	4	Очередь с приоритетом
из 2	∞	5	80	min(5,80) = 5
из 3	14		11	min(14,11) = 11
из 4				min(14) = 14
из 1				

Из 1-й вершины также никуда хода нет. Вычеркиваем ее:



Все вершины вычеркнуты. Найдены кратчайшие расстояния. Можно восстановить пути из второй вершины во все остальные. Дополнительно выпишем итог по второй строчке матрицы расстояний:

Текущая вершина	1	3	4
из 2	∞	5	80
из 3	14		11
из 4			
из 1			
Дошли	за 14	за 5	за 11
Предки	из 3-й	из 2-й	из 3-й

Ответы:

Из второй во все	Кратчайшие расстояния из второй вершины	Кратчайшие пути из второй вершины выписываем из итоговой таблицы.
2>1	14	Выписываем: 1-3-2 **Ответ. 2-3-1**
2>3	5	Выписываем: 3-2 **Ответ. 2-3**
2>4	11	Выписываем: 4-3-2 **Ответ. 2-3-4**

8.3 Программирование

Так как в программировании невозможно использовать понятие «бесконечность», вместо знака ∞ запишем максимально возможное число. В приведенном примере это будет число 1000.

```
program Dijkstra;
  // В графе n вершин:
  const n=4;
  // Бесконечность – infinity. В программе возьмем большое число – 1000
  const    infinity = 1000;

  // Исходный граф:
  const G: array[1..n, 1..n] of integer=
   ( (0,   1000,  1000, 1000),
     (1000,    0,      5,   80),
     (9,    1000,      0,    6),
     (1000,  17,  1000,     0) );

  var
    // Стартовая вершина:
    start: integer;
    // Вспомогательные переменные:
    count, index, i, u, m, min, pr, k,j: integer;
    // Массив расстояний:
    distance : array[1..n] of integer;
    // Массив для отметок посещенных вершин:
    visit    : array[1..n] of boolean;
    // Массив хранения предков для восстановления пути:
    predok   :  array [1..n] of integer;
    // Массив для хранения одного из путей:
    path : array [1..n] of integer;

BEGIN
  write('Стартовая вершина >> '); read(start);
  // Заполняем массив расстояний бесконечностью – 1000.
```

```
 for i:=1 to n do distance[i]:=infinity;

 for i:= 1 to n do  visit[i]:= false;     // Вершины не посещались.
 for i:= 1 to n do  predok[i]:=0;        // Пока у вершин предков нет.
 m:=start;                               // Запоминаем стартовую вершину.
 distance[start]:=0;

 for count:=1 to n-1 do begin            // Выполняем n – 1 шаг.

   // Ищем непосещенную вершину с минимальной меткой:
   min:=infinity;
   for i:=1 to n do
     if (not visit[i]) and (distance[i]<=min) then begin
       min:=distance[i]; index:=i;
     end;
   u:=index;                  // Запоминаем найденную вершину.

   // Вычеркиваем найденную вершину в массиве посещенных вершин:
   visit[u]:=true;

   // Пытаемся уменьшить метки у невычеркнутых соседей:
   for i:=1 to n do
     if (not visit[i]) and (G[u, i]<>0) and (distance[u]<>infinity) and
        (distance[u]+G[u, i]<distance[i]) then  begin
       distance[i]:=distance[u]+G[u, i];
       predok[i]:=u;            // Меняем предка у текущей вершины.
     end;
 end;

 // Выводим кратчайшее расстояние из стартовой вершины до всех:
 for i:=1 to n do
   if distance[i]<>infinity then begin
     writeln('Кратчайшее расстояние из ', m,' в ', i,' = ', distance[i]);
     j:=1;  path[j]:=i;
     while path[j] <> start do begin  // Пока предок – не стартовая вершина
       j:=j+1;
       path[j]:=predok[path[j-1]];    // Выписываем предка текущей вершины.
     end;

     // Выводим путь:
     writeln ( 'Кратчайший путь из ', start, ' в ', i );
     for k:= j downto 1 do write ( path[k], ' '  ); writeln;
   end
   else writeln(m,' > ', i,' = ', 'пути нет');

END.
```

8.4 Тестирование

Тест 1.

```
Стартовая вершина >> 1
Кратчайшее расстояние из 1 в 1 = 0
Кратчайший путь из 1 в 1
1
1 > 2 = пути нет
1 > 3 = пути нет
1 > 4 = пути нет
```

Тест 2.

```
Стартовая вершина >> 2
Кратчайшее расстояние из 2 в 1 = 14
Кратчайший путь из 2 в 1
2 3 1
Кратчайшее расстояние из 2 в 2 = 0
Кратчайшее расстояние из 2 в 3 = 5
Кратчайший путь из 2 в 3
2 3
Кратчайшее расстояние из 2 в 4 = 11
Кратчайший путь из 2 в 4
2 3 4
```

Тест 3.

```
Стартовая вершина >> 3
Кратчайшее расстояние из 3 в 1 = 9
Кратчайший путь из 3 в 1
3 1
Кратчайшее расстояние из 3 в 2 = 23
Кратчайший путь из 3 в 2
3 4 2
Кратчайшее расстояние из 3 в 3 = 0
Кратчайшее расстояние из 3 в 4 = 6
Кратчайший путь из 3 в 4
3 4
```

Тест 4.

```
Стартовая вершина >> 4
Кратчайшее расстояние из 4 в 1 = 31
Кратчайший путь из 4 в 1
4 2 3 1
Кратчайшее расстояние из 4 в 2 = 17
Кратчайший путь из 4 в 2
4 2
Кратчайшее расстояние из 4 в 3 = 22
Кратчайший путь из 4 в 3
4 2 3
Кратчайшее расстояние из 4 в 4 = 0
```

Контрольные вопросы

1 Сформулируйте понятие взвешенного графа и приведите примеры.

2 Назовите матрицу, с помощью которой задается структура взвешенного графа.

3 Сформулируйте основную идею алгоритма Дейкстры о поиске кратчайших путей из одной вершины во все остальные во взвешенном графе.

4 Что обозначает знак ∞ в матрице весов?

5 Как в программировании записывают «бесконечность»?

6 Сформулируйте понятие кратчайшего пути во взвешенном графе.

9 ПОИСК КРАТЧАЙШИХ ПУТЕЙ НА ВЗВЕШЕННОМ ГРАФЕ МЕЖДУ ВСЕМИ ПАРАМИ ВЕРШИН С ПОМОЩЬЮ АЛГОРИТМА ФЛОЙДА

9.1 Алгоритм Флойда

Рассмотрим следующие понятия.

1 Алгоритм Флойда как динамический алгоритм.

2 Поиск кратчайших путей с помощью рекуррентной формулы.

Алгоритм Флойда, также известный как алгоритм Флойда-Уоршелла, предназначен для поиска кратчайших расстояний и путей в транспортных системах между всеми парами вершин взвешенного графа. Граф может быть как ориентированным, так и неориентированным.

Этот алгоритм называется динамическим, потому что он последовательно вычисляет все значения элементов в матрице расстояний и пересчитывает это n раз, где n – количество вершин в графе. Для вычисления используют рекуррентную формулу

$$\begin{cases} d_{ij}^{(0)} = w_{ij}, \ \textit{где} \ \ w_{ij} - \textit{вес ребра} \ (i, j) \\ d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}, \ \textit{для} \ k \geq 1 \end{cases} \tag{1}$$

Рекуррентной формулой (от лат. *recurrere* – возвращаться) называют такую формулу, в которой каждый следующий элемент выражается через предыдущий элемент.

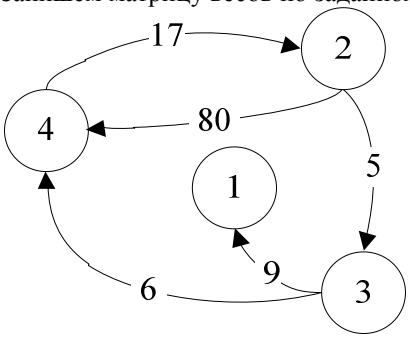
Задание 1.

ДАНО. Взвешенный орграф.

НАДО. Используя алгоритм Флойда, найти кратчайшие пути и расстояния между всеми парами вершин. Выписать последовательно все изменения в матрицах расстояний за время работы алгоритма.

Решение.

Запишем матрицу весов по заданному графу. Обозначим ее как $D^{(0)}$:



$D^{(0)} =$

	1	2	3	4
1	0	∞	∞	∞
2	∞	0	5	80
3	9	∞	0	6
4	∞	17	∞	0

Каждый элемент этой матрицы

$$d_{ij}^{(0)} = w_{ij}, \textit{где} \ w_{ij} - \textit{вес ребра} \ (i, j). \tag{2}$$

Вычеркнем 1-ю строку и 1-й столбец, так как в них указаны длины кратчайших путей без промежуточных вершин.

Далее матрицу весов будем использовать как **матрицу расстояний** и записывать в нее кратчайшие расстояния между двумя вершинами, если есть путь, их связывающий.

Алгоритм Флойда вычисляет итоговую матрицу расстояний посредством построения последовательности:

$D^{(0)}, D^{(1)}, \dots, D^{(k-1)}, D^{(k)}, \dots, D^{(n)}$.

Элемент d_{ij} в матрице на пересечении i-й строки и j-го столбца показывает длину кратчайшего пути от i-й вершины до j-й вершины.

Индекс k в рекуррентной формуле $d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$ показывает номер матрицы в последовательности.

Таким образом, вычисление каждого следующего элемента в матрице (k) сводится к сравнению текущего значения элемента и суммы элементов на пересечении i-й строки и j-го столбца в матрице ($k - 1$). Выбор будет сделан в сторону минимального из чисел.

$$d_{11}^{(1)} = \min\{d_{11}^{(0)}, d_{11}^{(0)} + d_{11}^{(0)}\} = \min\{0, 0+0\} = 0$$

$$d_{12}^{(1)} = \min\{d_{12}^{(0)}, d_{11}^{(0)} + d_{12}^{(0)}\} = \min\{\infty, 0+\infty\} = \infty$$

$$d_{13}^{(1)} = \min\{d_{13}^{(0)}, d_{11}^{(0)} + d_{13}^{(0)}\} = \min\{\infty, 0+\infty\} = \infty$$

$$d_{14}^{(1)} = \min\{d_{14}^{(0)}, d_{11}^{(0)} + d_{14}^{(0)}\} = \min\{\infty, 0+\infty\} = \infty$$

Вычислена первая строка матрицы расстояний.

$$d_{21}^{(1)} = \min\{d_{21}^{(0)}, d_{21}^{(0)} + d_{11}^{(0)}\} = \min\{\infty, \infty+0\} = \infty$$

$$d_{22}^{(1)} = \min\{d_{22}^{(0)}, d_{21}^{(0)} + d_{12}^{(0)}\} = \min\{0, \infty+\infty\} = 0$$

$$d_{23}^{(1)} = \min\{d_{23}^{(0)}, d_{21}^{(0)} + d_{13}^{(0)}\} = \min\{5, \infty+\infty\} = 5$$

$$d_{24}^{(1)} = \min\{d_{24}^{(0)}, d_{21}^{(0)} + d_{14}^{(0)}\} = \min\{80, \infty+\infty\} = 80$$

Вычислена вторая строка матрицы расстояний.

Так же будут вычислены остальные строки.

$$d_{31}^{(1)} = \min\{d_{31}^{(0)}, d_{31}^{(0)} + d_{11}^{(0)}\} = \min\{9, 9+0\} = 9$$

$$d_{32}^{(1)} = \min\{d_{32}^{(0)}, d_{31}^{(0)} + d_{12}^{(0)}\} = \min\{\infty, 9+\infty\} = \infty$$

$$d_{33}^{(1)} = \min\{d_{33}^{(0)}, d_{31}^{(0)} + d_{13}^{(0)}\} = \min\{0, 9+\infty\} = 0$$

$$d_{34}^{(1)} = \min\{d_{34}^{(0)}, d_{31}^{(0)} + d_{14}^{(0)}\} = \min\{6, 9+\infty\} = 6$$

$$d_{41}^{(1)} = \min\{d_{41}^{(0)}, d_{41}^{(0)} + d_{11}^{(0)}\} = \min\{\infty, \infty + 0\} = \infty$$

$$d_{42}^{(1)} = \min\{d_{42}^{(0)}, d_{41}^{(0)} + d_{12}^{(0)}\} = \min\{17, \infty + \infty\} = 17$$

$$d_{43}^{(1)} = \min\{d_{43}^{(0)}, d_{41}^{(0)} + d_{13}^{(0)}\} = \min\{\infty, \infty + \infty\} = \infty$$

$$d_{44}^{(1)} = \min\{d_{44}^{(0)}, d_{41}^{(0)} + d_{14}^{(0)}\} = \min\{0, \infty + \infty\} = 0$$

Итоги представлены в таблице $D^{(1)}$.

$D^{(1)} =$

	1	2	3	4
1	0	∞	∞	∞
2	∞	0	5	80
3	9	∞	0	6
4	∞	17	∞	0

Результат можно пояснить так: выписаны длины кратчайших путей с промежуточными вершинами, номера которых не больше 1. То есть здесь искали кратчайшие пути, которые проходили через вершину 1. В данном случае таких путей не оказалось, и в таблице не было произведено изменений.

Такие вычисления следует провести и для всех остальных таблиц.

$D^{(2)} =$

	1	2	3	4
1	0	∞	∞	∞
2	∞	0	5	80
3	9	∞	0	6
4	∞	17	**22**	0

Нашли длины кратчайших путей с промежуточными вершинами, номера которых не больше 2, то есть 1 и 2. Новый кратчайший путь от 4-й к 3-й вершине.

$D^{(3)} =$

	1	2	3	4
1	0	∞	∞	∞
2	**14**	0	5	**11**
3	9	∞	0	6
4	**31**	17	22	0

Новые кратчайшие пути через 3 вершины, то есть 1, 2 и 3. Новые кратчайшие пути:

от 2-й к 1-й вершине;

от 2-й к 4-й вершине;

от 4-й к 1-й вершине.

$D^{(4)} =$

	1	2	3	4
1	0	∞	∞	∞
2	14	0	5	11
3	9	**23**	0	6
4	31	17	22	0

На последнем этапе доступны все вершины. Найден новый кратчайший путь от 3-й ко 2-й вершине.

В матрице $D^{(4)}$ записаны кратчайшие расстояния между всеми парами вершин взвешенного графа

9.2 Программирование: поиск кратчайших расстояний

Так как в программировании невозможно использовать понятие «бесконечность», вместо знака ∞ запишем максимально возможное число. В приведенном примере это будет число 1000.

```
program Floid;
  // В графе n вершин:
  const n=4;
  // Исходный граф как матрица весов:
  const Graph:     array [1..n,1..n] of integer =
  ((  0, 1000, 1000, 1000),
   (1000,   0,    5,   80),
   (  9, 1000,    0,    6),
   (1000,  17, 1000,    0) );
  // Бесконечность – infinity. В программе возьмем большое число – 1000
  const    infinity = 1000;
  // Матрица кратчайших расстояний между двумя вершинами:
  var G : array[1..n, 1..n] of integer;
  // Вспомогательные переменные:
  var i, j,  inf, k: integer;

BEGIN
  writeln('Матрица весов=');
  for i:=1 to n do begin
    for j:=1 to n do
      write(Graph[i, j]:6);
    writeln;
  end;
  // Передаем данные матрицы весов в матрицу расстояний:
  G:=Graph;
  // Ищем по формуле кратчайшие расстояния между двумя вершинами:
  for k:=1 to n do begin
    writeln('Матрица кратчайших расстояний ', k);
    for i:=1 to n do
      for j:=1 to n do
        if (G[i, k]+G[k, j]<G[i, j])and (i<>j) then
          G[i, j]:=G[i, k]+G[k, j];
    // Выводим промежуточные и итоговую матрицы кратчайших
расстояний:
    for i:=1 to n do begin
      for j:=1 to n do
        write(G[i, j]:6);
      writeln;
    end;
  end;
END.
```

9.3 Тестирование: поиск кратчайших расстояний

```
  Матрица весов=
     0  1000  1000  1000
  1000     0     5    80
     9  1000     0     6
  1000    17  1000     0
Матрица кратчайших расстояний 1
     0  1000  1000  1000
  1000     0     5    80
     9  1000     0     6
  1000    17  1000     0
Матрица кратчайших расстояний 2
     0  1000  1000  1000
  1000     0     5    80
     9  1000     0     6
  1000    17    22     0
Матрица кратчайших расстояний 3
     0  1000  1000  1000
    14     0     5    11
     9  1000     0     6
    31    17    22     0
Матрица кратчайших расстояний 4
     0  1000  1000  1000
    14     0     5    11
     9    23     0     6
    31    17    22     0
```

9.4 Программирование: поиск кратчайших расстояний и путей

```
program Floid;
  // В графе n вершин:
  const n=4;
  // Исходный граф как матрица весов:
  const Graph:     array [1..n,1..n] of integer =
  ((  0, 1000, 1000, 1000),
   (1000,   0,    5,   80),
   (  9, 1000,    0,    6),
   (1000,  17, 1000,    0) );
  // Бесконечность – infinity. В программе возьмем большое число – 1000
  const    infinity = 1000;
  // Матрица кратчайших расстояний между двумя вершинами:
  var G : array[1..n, 1..n] of integer;
  // Вспомогательные переменные:
  var i, j,  inf, k, p: integer;
  // Массив хранения предков для восстановления пути:
  predok   :  array [1..n, 1..n] of integer;
  // Массив для хранения одного из путей:
  path : array [1..n] of integer;
BEGIN
```

```
  writeln('Матрица весов=');
  for i:=1 to n do begin
    for j:=1 to n do write(Graph[i, j]:6);
    writeln;
  end;
  // Передаем данные матрицы весов в матрицу расстояний:
  G:=Graph;
  // Заполняем массив предков номерами стартовых вершин:
  for i:=1 to n do
    for j:=1 to n do  predok[i,j]:=i;
  // Ищем по формуле кратчайшие расстояния между двумя вершинами:
  for k:=1 to n do begin
    writeln('Матрица кратчайших расстояний ', k);
    for i:=1 to n do
      for j:=1 to n do
        if (G[i, k]+G[k, j]<G[i, j])and (i<>j) then begin
          G[i, j]:=G[i, k]+G[k, j];  // Вычисляем кратчайшее расстояние.
          predok[i,j]:=predok[k,j];  // Меняем предка у текущей вершины.
        end;
    // Выводим промежуточные и итоговую матрицы кратчайших расстояний:
    for i:=1 to n do begin
      for j:=1 to n do write(G[i, j]:6);
      writeln;
    end;
  end;

 // Выводим предков, т. е. предпоследнюю вершину в пути от i к j
  writeln('Предки вершин, через которые проходит путь ');
  for i:=1 to n do begin
    for j:=1 to n do   Write(predok[i,j],' ' );
    writeln;
  end;

  writeln('Кратчайшие пути между всеми парами вершин:');
  for i:=1 to n do
    for j:=1 to n do
      if  (G[i,j]<>infinity) and(i<>j) then begin
        p:=1;  path[p]:=j;
        while path[p] <> i do begin // Пока предок не равен стартовой вершине.
          p:=p+1;
          path[p]:=predok[i,path[p-1]]; //Выписываем предка текущей вершины.
        end;

// Выводим пути:
        write ( 'Кратчайший путь из ', i, ' в ', j ,' = ');
        for k:= p downto 1 do write ( path[k], ' '  ); writeln;
```

```
    end
    else writeln('Кратчайший путь из ', i, ' в ', j ,' = ', 'пути нет');
END.
```

9.5 Тестирование: поиск кратчайших расстояний и путей

```
  Матрица весов=
     0  1000  1000  1000
  1000     0     5    80
     9  1000     0     6
  1000    17  1000     0
Матрица кратчайших расстояний 1
     0  1000  1000  1000
  1000     0     5    80
     9  1000     0     6
  1000    17  1000     0
Матрица кратчайших расстояний 2
     0  1000  1000  1000
  1000     0     5    80
     9  1000     0     6
  1000    17    22     0
Матрица кратчайших расстояний 3
     0  1000  1000  1000
    14     0     5    11
     9  1000     0     6
    31    17    22     0
Матрица кратчайших расстояний 4
     0  1000  1000  1000
    14     0     5    11
     9    23     0     6
    31    17    22     0
Предки вершин, через которые проходит путь
1 1 1 1
3 2 2 3
3 4 3 3
3 4 2 4
Кратчайшие пути между всеми парами вершин:
Кратчайший путь из 1 в 1 = пути нет
Кратчайший путь из 1 в 2 = пути нет
Кратчайший путь из 1 в 3 = пути нет
Кратчайший путь из 1 в 4 = пути нет
Кратчайший путь из 2 в 1 = 2 3 1
Кратчайший путь из 2 в 2 = пути нет
Кратчайший путь из 2 в 3 = 2 3
Кратчайший путь из 2 в 4 = 2 3 4
Кратчайший путь из 3 в 1 = 3 1
Кратчайший путь из 3 в 2 = 3 4 2
Кратчайший путь из 3 в 3 = пути нет
Кратчайший путь из 3 в 4 = 3 4
Кратчайший путь из 4 в 1 = 4 2 3 1
Кратчайший путь из 4 в 2 = 4 2
Кратчайший путь из 4 в 3 = 4 2 3
Кратчайший путь из 4 в 4 = пути нет
```

Контрольные вопросы

1 Назовите матрицу, с помощью которой задается структура взвешенного графа.

2 Сформулируйте основную идею алгоритма Флойда о поиске кратчайших путей между всеми парами вершин во взвешенном графе.

10 СТРУКТУРА И ОФОРМЛЕНИЕ КОНТРОЛЬНОЙ ИЛИ РАСЧЕТНО-ГРАФИЧЕСКОЙ РАБОТЫ

Контрольная работа должна содержать следующие структурные части:

1) титульный лист – является первым листом работы, но номер на нем не ставится;

2) задание, выданное преподавателем, – в нумерацию листов не включается, если лист задания выдан преподавателем в распечатанном виде;

3) содержание – должно быть создано автоматически на основе стилей типа «Заголовок»;

4) перечень условных обозначений (при необходимости);

5) введение – говорится об актуальности работы, какую цель она преследует и какие задачи решаются для достижения этой цели; объем 1–1,5 с.;

6) основную часть, разбитую на разделы;

7) заключение – приводят выводы из работы, а не общие слова (как это важно); фактически дают ответы на те задачи, которые были поставлены во введении. Объем 1–2 с.;

8) список использованных источников – должен включать до 10 наименований печатных и электронных источников. Список формируют в алфавитном порядке фамилий первых авторов и (или) заглавий;

Пример оформления списка.

1 Гораев, О. П. Информатика. Математические и физические основы ЭВМ : учеб. пособие / О. П. Гораев, Т. Н. Модина. – Гомель : БелГУТ, 2007. – 31 с.

2 Миняйлова, Е. Л. Решение инженерных задач средствами информационных компьютерных комплексов: задача «Логические элементы в системах управления на транспорте» : учеб.-метод. пособие / Е. Л. Миняйлова. – Гомель : БелГУТ, 2012. – 37 с.

3 Национальный Интернет-портал Республики Беларусь [Электронный ресурс] / Нац. центр правовой информ. Респ. Беларусь. – Минск, 2005. – Режим доступа : http://www.pravo.by. – Дата доступа : 25.01.2006.;

9) приложения (при необходимости).

Страницы нумеруют в центре нижней части листа.

Каждую структурную часть следует начинать с нового листа.

К защите работы необходимо представить в электронном виде файлы с именем:

Группа_Фамилия_Имя.pas
RGR.dot
Группа_Фамилия_Имя.doc

Также предоставляются изображение графа, карта, с которой получено изображение графа, другие рабочие материалы в электронном виде.

В итоговом документе оформление должно быть выполнено с помощью стилей.

11 СТИЛЕВОЕ ОФОРМЛЕНИЕ ДОКУМЕНТА В ТЕКСТОВОМ ПРОЦЕССОРЕ

11.1 Основные понятия

Для правильного оформления текстового документа следует выделить логическую структуру и уточнить смысловое назначение элементов текста, на которых нужно акцентировать внимание.

Логическая структура текста получается разделением его на главы, разделы, подразделы, пункты, абзацы текста.

Разделы, подразделы, пункты и подпункты нумеруются арабскими цифрами с точками и записываются с нового абзаца.

Номер подраздела включает номера раздела и подраздела, разбитые точкой (1.1, 1.2, 1.3 и т. д.). Номер пункта – номера раздела, подраздела и пункта, разбитые точками (1.1.1, 1.1.2, 1.1.3 и т. д.). Номер подпункта – номера раздела, подраздела, пункта и подпункта, разбитые точками (1.2.3.4, 1.2.3.5 и т. д.).

Разделы должны иметь заголовки. Подразделы при необходимости также могут иметь заголовки. В заголовках не допускается перенос слов, применение римских цифр, математических знаков и греческих букв. Точка в конце заголовка не ставится.

Назначение элементов текста – их роль в тексте: определение термина, формула, заголовок таблицы, подрисуночная подпись. А за внешний вид этих элементов текста в текстовом процессоре будет отвечать стиль. Например, определение должно быть выделено полужирным начертанием шрифта, а формула – размещена с выравниванием по центру.

Таким образом, смысловое назначение элементов текста является основанием для назначения стиля.

Стиль – это инструмент, помогающий быстро форматировать документ. Под стилем понимают специально созданные наборы форматирования, позволяющие использовать одновременно несколько атрибутов. Стилям присваивают имена.

Различают два типа стилей: абзаца и символа.

Стиль абзаца применяется целиком к абзацу. Этот стиль определяет как тип и размер шрифта, так и размещение абзаца, интервалы для всего текста. В области задач стиль абзаца помечается значком ¶.

Стиль символа (знака) применяется в дополнение к стилю абзаца к отдельному символу или последовательности символов. Стили знаков помечаются в области задач значком a.

Шаблон – это файл с расширением dot, на основе которого создаются документы. В шаблон сохранены необходимые стили.

Посмотреть, с каким шаблоном вы сейчас работаете, можно так: «**Сервис – Шаблоны и надстройки»** (рисунок 37).

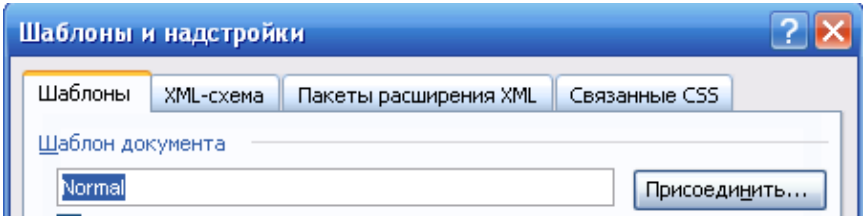


Рисунок 37 – Фрагмент диалогового окна «Шаблоны и надстройки»

11.2 Создание шаблона и документа для контрольной или расчетно-графической работы

В папке, например, «Расчетно-графическая работа_Группа_ Фамилия_Имя» создайте новый текстовый документ с именем «Группа_ Фамилия_Имя», например, СП21_Иванов_Иван.doc. Это можно сделать разными способами.

Способ 1. На основе шаблона Обычный. По умолчанию документы Word создают на базе обычного шаблона Normal.dot.

В шаблоне Normal.dot сохраняют все стили по умолчанию, поэтому лучше всего оставить его неизменным, а для расчетно-графической работы использовать переименованную копию Normal.dot.

Способ 2. На основе своего шаблона RGR.dot.

Шаг 1. Создайте свой шаблон RGR.dot для расчетно-графической работы с помощью последовательности действий:

1) создать новый документ;

2) дать команду «**Файл – Сохранить как…**». Найти свою папку, в списке «**Тип файла**» внизу окна выбрать «**Шаблон документа**», ввести имя файла RGR, расширение **dot** не менять, сохранить файл (рисунок 38). Вновь созданный шаблон является переименованной копией Normal.dot. Дальше можно изменять нужные элементы.

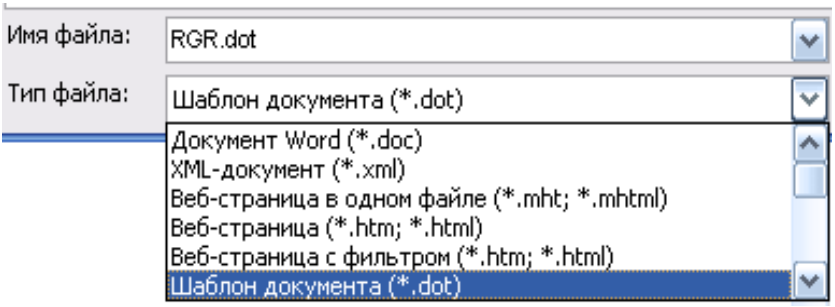


Рисунок 38 – Сохранение нового шаблона

Шаг 2. Создайте новый документ на основе шаблона RGR.dot. Откройте папку с сохраненным шаблоном. Дважды щелкните мышью на его имени, при этом откроется не сам шаблон, а новый документ, созданный на его базе. Сохраните документ с именем **Группа_Фамилия_Имя**, например, **СП21_Иванов_Иван.doc.** Продолжайте работу в этом документе.

Укажите основные настройки для нового шаблона. Установите параметры страницы (рисунок 39).

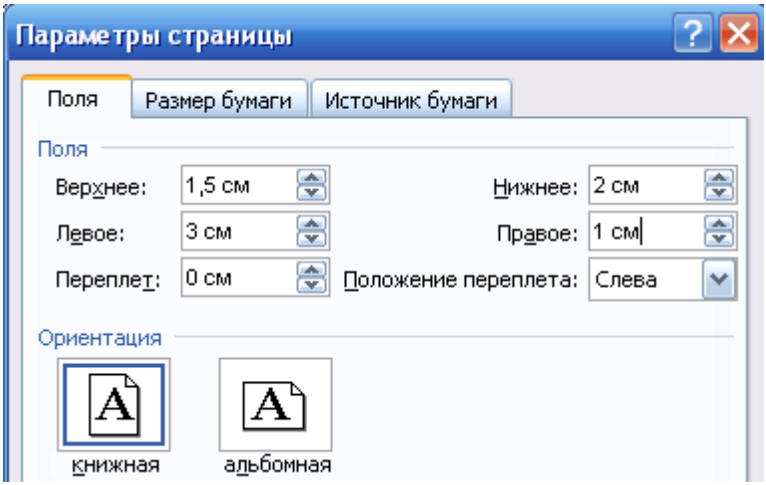


Рисунок 39 – Параметры страницы

11.3 Создание стилей для контрольной или расчетно-графической работы

Командой **Формат – Стили и форматирование** открываем окно «**Стили**» (рисунок 40).

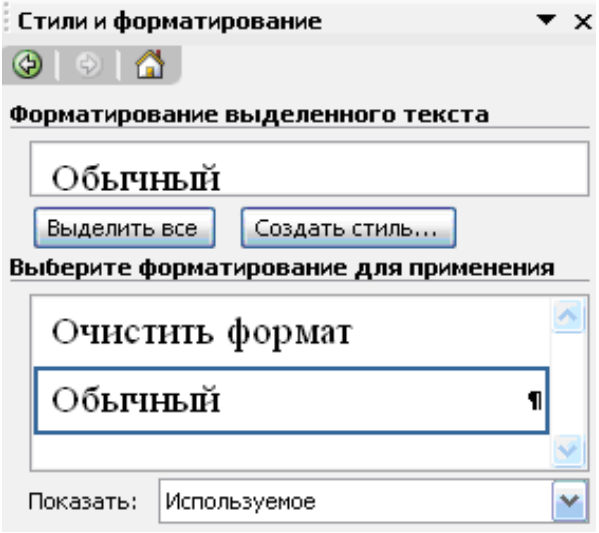


Рисунок 40 – Окно «Стили» на панели «Форматирование»

Настройте стиль абзаца на основе стандартного стиля «Обычный» (рисунки 41, 42).

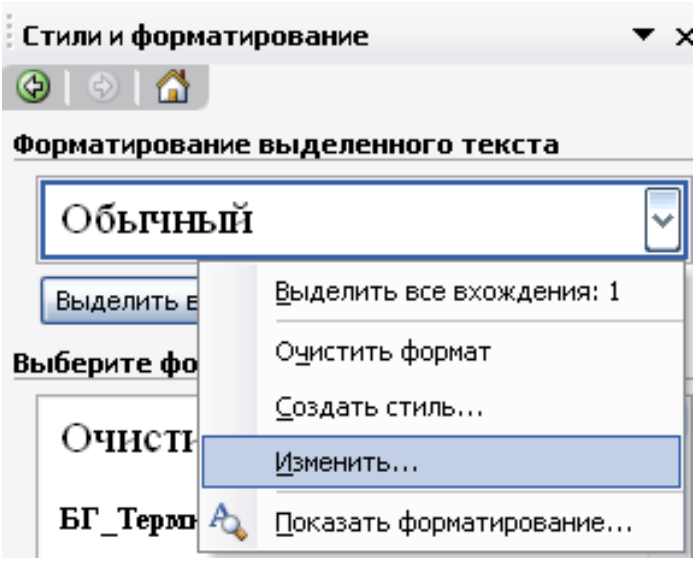


Рисунок 41 – Изменение стиля «Обычный»

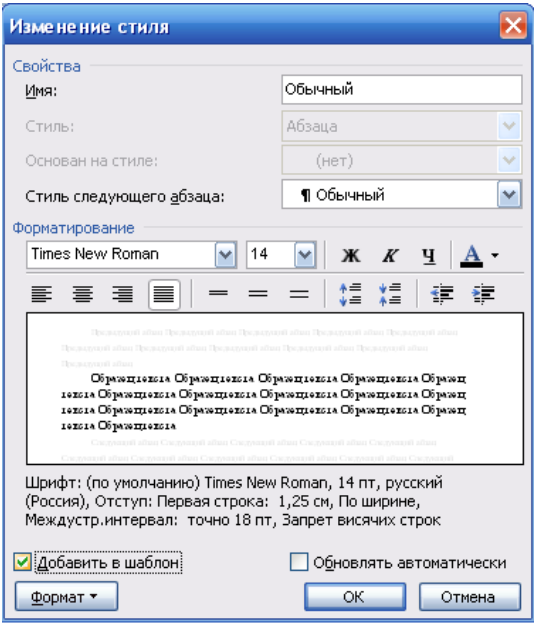


Рисунок 42 – Настройка стиля «Обычный»

Параметры для основного абзаца устанавливают после нажатия кнопки «Формат – Абзац» в окне «Изменение стиля» (рисунок 43).

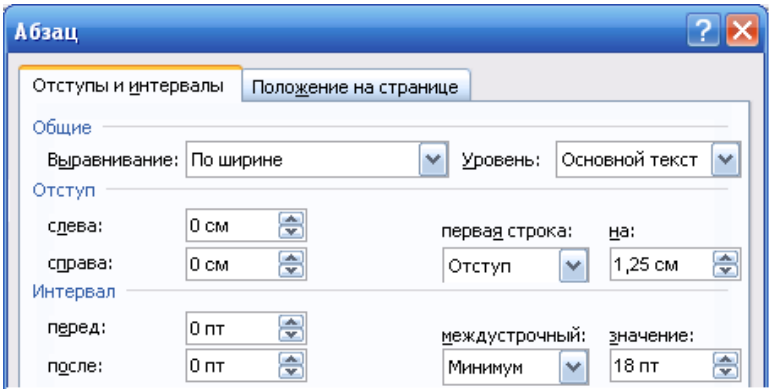


Рисунок 43 – Настройка параметров абзаца

После настройки стандартного стиля «Обычный» можно создать на его основе новые стили. Для этого достаточно щелкнуть кнопку «Создать стиль» (рисунок 44).

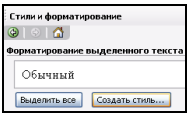


Рисунок 44 – Создание нового стиля

При создании нового стиля требуется указать его имя (например, Р_Министерство), тип стиля (абзац или знак), на каком стиле основан новый стиль (например, на стиле «Обычный»), стиль следующего абзаца (рисунок 45). Также требуется указать все элементы форматирования: шрифт, абзац, границы. Новый стиль рекомендуется добавить в шаблон.

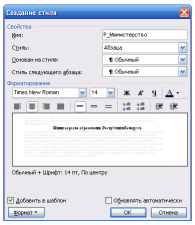


Рисунок 45 – Создание стиля Р_Министерство на основе стиля «Обычный»

Выпишем, какие стили понадобятся для титульного листа (таблица 1).

Таблица 1 **– Форматирование титульного листа**

Имя стиля	Тип стиля	Основан на стиле	Стиль следующего абзаца	Отличие форматирования от стиля «Обычный»
Р_Министерство	Абзац	Обычный	Обычный	**Абзац:** Выравнивание – по центру. Уровень – основной текст. Отступ: слева – 0, справа – 0, первая строка – отступа нет, интервал перед – 0, после – 30, междустрочный – минимум 18 пт
Р_Учреждение	Абзац	Обычный	Обычный	**Абзац:** Выравнивание – по центру, первая строка – отступа нет
Р_Кафедра	Абзац	Обычный	Обычный	**Абзац:** Выравнивание – по центру, первая строка – отступа нет, интервал перед – 24, после – 0
Р_РГР	Абзац	Обычный	Обычный	**Абзац:** Выравнивание – по центру, первая строка – отступа нет, интервал перед – 102, после – 0
Р_Тема	Абзац	Обычный	Обычный	**Шрифт:** Размер – 20. **Абзац:** Выравнивание – по центру, первая строка – отступа нет, интервал перед – 30, после – 132
Р_Выполнил	Абзац	Обычный	Обычный	**Шрифт:** Размер – 14. **Абзац:** Выравнивание – по ширине, первая строка – отступа нет, интервал перед – 0, после – 0, позиции табуляции: 10,75 см, по левому краю

Окончание таблицы 1

Имя стиля	Тип стиля	Основан на стиле	Стиль следующего абзаца	Отличие форматирования от стиля «Обычный»
Р_Гомель	Абзац	Обычный	Обычный	**Шрифт:** Размер – 14. **Абзац:** Выравнивание – по центру, первая строка – отступа нет, интервал перед – 150, после – 0

Как видно из пояснений, стиль «Обычный» не применяется при наборе текста. Он используется как основа для построения стилей.

При необходимости можно создать свои стили. **Основанием для назначения стиля является смысловое назначение элементов текста!**

Результат выполнения работ по созданию стилей для титульного листа и набора соответствующего текста приведен на рисунке 46.

Рисунок 46 – Стилевое оформление титульного листа

Стили для оформления текста работы приведены в таблице 2.

Таблица 2 **– Форматирование текста расчетно-графической работы**

Имя стиля	Тип стиля	Основан на стиле	Стиль следующего абзаца	Отличие форматирования от стилей «Обычный» и «Заголовок»
Р_Главный	Абзац	Обычный	К_Главный	
Р_Подрисуночный	Абзац	Обычный	К_Главный	**Шрифт:** Размер – 12. **Абзац:** Выравнивание – по центру. Уровень – основной текст. Отступ: слева 0, справа 0, первая строка – отступа нет, интервал перед – 6, после – 4, не разрывать абзац
Р_Рисунок	Абзац	Обычный	К_Подрисуночный	**Абзац:** Выравнивание – по центру, первая строка – отступа нет, не отрывать от следующего
Р_Термин	Знак	Обычный	К_Главный	**Шрифт:** полужирный
Р_Решение	Знак	Обычный	К_Главный	**Шрифт:** полужирный, курсив
Р_Паскаль	Абзац	Обычный	К_Паскаль	**Шрифт:** Verdana, английский (США) **Шрифт:** Размер – 10. **Абзац:** первая строка – отступа нет
Р_Содержание	Абзац	Обычный	К_Главный	**Абзац:** Выравнивание – по центру. Уровень – основной текст. Отступ: первая строка – отступа нет, интервал перед – 6, после – 4, не разрывать абзац
Р_Список нумерованный	Абзац	Обычный	К_Список нумерованный	Создать список, а на его основе создать стиль
Р_Список маркированный	Абзац	Обычный	К_Список маркированный	Создать список, а на его основе создать стиль
Р_Табл_Название	Абзац	Обычный	К_Табл_Название	**Абзац:** Выравнивание – по левому краю. Уровень – основной текст. Отступ: первая строка – отступа нет, интервал перед – 6, после – 4, не разрывать абзац
Р_Табл_Загол_столбцов	Абзац	Обычный	К_Табл_Загол_столбцов	**Абзац:** Выравнивание – по центру. Уровень – основной текст. Отступ: первая строка – отступа нет, интервал перед – 0, после – 0
Р_Заголовок 2	Абзац	Заголовок 2	К_Главный	Уровень – 2

Окончание таблицы 2

Имя стиля	Тип стиля	Основан на стиле	Стиль следующего абзаца	Отличие форматирования от стилей «Обычный» и «Заголовок»
Р_Заголовок 1	Абзац	Заголовок 1	К_Главный	Уровень – 1. Положение на странице – с новой страницы (рисунок 47)
Р_Табл_Осн_ Текст	Абзац	Обычный	К_Табл_Осн_ Текст	**Шрифт:** Размер – 13. **Абзац:** Выравнивание – по левому краю. Отступ: первая строка – отступа нет, интервал перед – 0, после – 0, не разрывать абзац

Порядок оформления заголовка 1-го уровня с новой страницы представлен на рисунке 47.

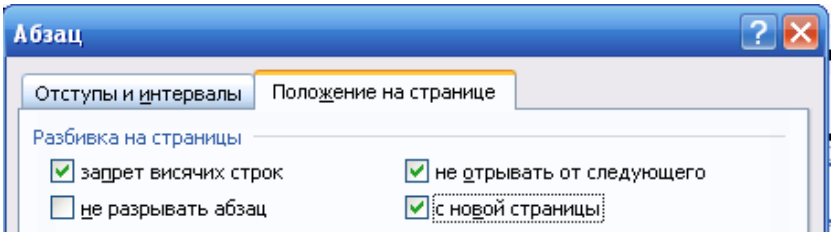


Рисунок 47 – Оформление заголовка 1-го уровня с новой страницы

11.4 Автоматическое создание оглавлений

Содержание должно быть создано автоматически на основе стилей типа «Заголовок». Для этого требуется выполнить следующие действия.

1 Установить курсор после слова СОДЕРЖАНИЕ.

2 Дать команду «Вставка» – «Ссылка» – «Оглавления и указатели». Выбрать вкладку «Оглавления» и нажать ОК.

3 Если потребуется обновить оглавление, то это можно сделать клавишей F9 или командой контекстного меню «Обновить поле».

Контрольные вопросы

1 Что является основанием для назначения стиля элементам текста?

2 Назовите расширение файла у шаблона, на основе которого создаются документы.

4 На основе каких стилей можно автоматически создать оглавление?

СПИСОК ИСПОЛЬЗОВАННОЙ И РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1 **Ахо, А. В.** Структуры данных и алгоритмы / А. В. Ахо, Дж. Э. Хопкрофт, Дж. Д. Ульман. – М. : Изд. дом «Вильямс», 2000. – 384 с.

2 **Вирт, Н.** Алгоритмы и структуры данных / Н. Вирт. – СПб. : Невский Диалект, 2001. – 352 с.

3 **Евстигнеев, В. А.** Применение теории графов в программировании / В. А. Евстигнеев ; под ред. А. П. Ершова. – М. : Наука. Гл. ред. физ.-мат. лит., 1985. – 352 с.

4 Изучение алгоритма поиска в глубину с помощью структуры данных стек / Е. Л. Миняйлова [и др.] // Информатика и образование. – 2011. – № 2. – С. 35–39.

5 **Калачик, Р. А.** Методы поиска графической информации в информационных системах: дис. ... канд. техн. наук : 05.13.11 / Р. А. Калачик; Тул. гос. ун-т. – Тула, 2008. – 161 с. : ил.

6 **Кормен, Т.** Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М. : МЦНМО, 1999. – 960 с.

7 **Котов, В. М.** Структуры данных и алгоритмы: теория и практика / В. М. Котов, Е. П. Соболевская. – Минск : БГУ, 2004. – 255 с.

8 **Миняйлова, Е. Л.** Решение инженерных задач средствами информационных компьютерных комплексов: задача «Логические элементы в системах управления на транспорте» : учеб.-метод. пособие / Е. Л. Миняйлова. – Гомель : БелГУТ, 2012. – 37 с.

Учебное издание

МИНЯЙЛОВА Елена Леонидовна
ВЕРБОВИКОВ Дмитрий Александрович
КОЛЕДА Наталья Ремовна
МИНЯЙЛОВ Владимир Сергеевич

ИНФОРМАТИКА И КОМПЬЮТЕРНОЕ ПРОЕКТИРОВАНИЕ

Учебно-методическое пособие

Редактор *Н. Г. Шеметкова*
Технический редактор *В. Н. Кучерова*
Корректор *А. А. Павлюченкова*

Подписано в печать 14.12.2015 г. Формат 60×84 1/16
Бумага офсетная. Гарнитура Times. Печать на ризографе.
Усл. печ. л. 3,95. Уч.-изд. л. 3,93 . Тираж 150 экз.
Зак. № 4299. Изд. № 36.

Издатель и полиграфическое исполнение:
Белорусский государственный университет транспорта.
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/361 от 13.06.2014
№ 2/104 от 01.04.2014
Ул. Кирова, 34, 246653, г. Гомель