Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Исследование алгоритмов поиска на графах

БГУИР КП I–40 01 01 323 ПЗ

Выполнил

студент гр. 051003 Павловец С.В.

Проверил: Фадеева Е.П.

Минск 2021

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Лапицкая Н.В. 2021г.

ЗАДАНИЕ

по курсовому проектированию

Студенту *Павловцу Сергею Валерьевичу*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Тема работы *Исследование алгоритмов поиска на графах\_\_\_\_\_\_\_\_\_\_\_\_\_*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Срок сдачи законченной работы *11.06.2021г.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Исходные данные к работе *Среда программирования Delphi. Реализация функций создания и редактирования графов. Возможность сохранения созданных графов в удобном формате, открытия файлов с графами, сохраненными в формате программы. Возможность отображения графа на экране, экспорта изображения графа в файл формата SVG. Реализация функций демонстрации работы алгоритмов поиска на созданных графах. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*1 Анализ литературных источников\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2 Постановка задачи\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*3 Разработка программного средства\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*4 Тестирование и проверка работоспособности программного средства\_\_\_\_*

*5 Руководство по установке и использованию программного средства\_\_\_\_\_*

*Заключение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Список использованных источников\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

\_*Приложения* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*Схема алгоритма в формате А1*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Консультант по курсовой работе *Фадеева Е.П.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

7.Дата выдачи задания *16.02.2021г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего обьема работы):

*Раздел 1. Введение к 28.02.2021г. – 10 % готовности работы;\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 2 к 15.03.2021г. – 30% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 3 к 15.04.2021г. – 60% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 4 к 10.05.2021г. – 80% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 5.Заключение. Приложения к 20.05.2021г. – 90% готовности работы;*

*оформление пояснительной записки и графического материала к 01.06.2021г. – 100% готовности работы.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Защита курсового проекта с 02.06.2021г. по 11.06.2021г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

РУКОВОДИТЕЛЬ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Фадеева Е.П.*

*(подпись)*

Задание принял к исполнению *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Павловец С.В. 16.02.2021г.*

*(дата и подпись студента)*

Содержание

[Введение 5](#_Toc67352020)

[1 Анализ литературных источников 6](#_Toc67352021)

[1.1 Понятие графа и его представление в памяти компьютера 6](#_Toc67352022)

[1.2 Структура данных «однонаправленный список» 7](#_Toc67352023)

[1.3 Обходы графов в глубину и ширину 8](#_Toc67352024)

[1.4 Алгоритм Дейкстры для взвешенных графов 10](#_Toc67352025)

[1.5 Работа с текстовыми файлами 12](#_Toc67352026)

[2 Постановка задачи 16](#_Toc67352027)

# Введение

Данный курсовой проект посвящен исследованию алгоритмов поиска на графах.

Теория графов – раздел дискретной математики, изучающий такие математические объекты, как графы. Её родоначальником считается Леонард Эйлер, который в 1736 году формулирует и предлагает решение задачи о семи кёнигсбергских мостах. Сам термин «граф» был предложен в 1878 году Сильвестром Джеймсом Джозефом. Теория графов развивалась в процессе решения загадок и головоломок, так как графы наглядно представляются графически с помощью фигур и соединяющих их линий. И хотя долгое время она не выделялась как отдельная математическая дисциплина, но в 20-21-ые столетия графы начались использоваться во многих сферах будь то гуманитарные науки или технические. Графы также активно применяются в программировании для описания различных алгоритмов, структур данных, синтаксиса языков и т.д.

Важной частью теории графов являются задачи поиска. К примеру, согласно формулировке задачи, может быть необходимо найти некую вершину, начиная просмотр из заданной, или же некий путь, удовлетворяющий определённым условиям, или какой-либо подграф заданного графа, обладающий указанными свойствами.

Программное средство предназначено для создания и редактирования графов, а также для демонстрации некоторых алгоритмов поиска.

Целью работы является создание проекта, сопровождающегося документацией в виде пояснительной записки.

В реализации проекта попутно решаются следующие задачи:

* создание динамических структур;
* работа с файлами (текстовыми, типизированными);
* чтение/запись данных из файла;
* разработка пользовательского интерфейса для реализации перечисленных функций;
* представление содержимого файла для работы в виде динамической структуры: однонаправленного списка.

# Анализ литературных источников

## Понятие графа и его представление в памяти компьютера

Простым (n, m)-графом называется пара конечных множеств вершин и рёбер таких, что множество вершин состоит из n>0 элементов, а множество рёбер - из m элементов, каждый из которых является неупорядоченной парой из вершин. Если же помимо перечисленных условий каждому ребру ставится в соответствие некое число (вес), то такие множества называют взвешенным графом. Существует несколько способов задания графов в памяти компьютера.

Первый из них – матрица смежности. У неё номерам столбцов и строк соответствуют номера вершин графа. В каждой ячейке записывается число, определяющее смежность вершин: если вершины смежны, то элемент равен 1, иначе – 0. Для взвешенных графов вместо матрицы смежности используется матрица весов, где вместо нулей и единиц записывают вес ребра, если же ребра не существует, то в элемент записывают 0 или ∞ в зависимости от задачи. Подобный способ представления занимает O(n2) места в памяти.

Следующий способ – списками смежности. В этом случае граф представляет собой множество окрестностей всех вершин. Окрестность вершины, в свою очередь, представляется множеством смежных с ней вершин. Для взвешенного графа вместе с номером смежной вершины хранится вес соответствующего ребра. Таким образом, подобная структура занимает места в памяти.

Видно, что для графов с относительно небольшим количеством рёбер (их называют разреженными) меньше места в памяти занимает второй способ представления. Если же рёбер много (такие графы называют плотными), в частности для полного графа, у которого , списки смежности могут занимать больше памяти, чем матрица смежности. Стоит также учесть, что временная сложность проверки смежности вершин в матрице смежности составляет O(1), а в списках смежности – O(n).

Существуют и такие способы задания графов, как матрица инцидентности, строки которой соответствуют вершинам, а столбцы – рёбрам, и список рёбер, который является перечислением каждого ребра графа. Однако, они реже используются для решения задач на графах: матрица инцидентности имеет размер O(n\*m), а временная сложность проверки смежности вершин в списке рёбер – O(m), что хуже, чем у списка смежности для плотных графов. Более того, список рёбер не полностью описывает графы с изолированными вершинами, поэтому чтобы необходимо хранить дополнительный список вершин.

В следующих таблицах приведены примеры различного представления графа, изображённого на рисунке 1.

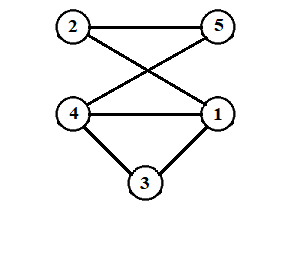


Рисунок 1.1 - Пример графа

Таблица 1 - Матрица смежности

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

Таблица 2 - Списки смежности

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **→** | 2 | 3 | 4 |
| **2** | **→** | 1 | 5 |  |
| **3** | **→** | 1 | 4 |  |
| **4** | **→** | 1 | 3 | 5 |
| **5** | **→** | 2 | 4 |  |

Таблица 3 - Матрица инцидентности

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **{1, 2}** | **{1, 3}** | **{1, 3}** | **{2, 5}** | **{3, 4}** | **{4, 5}** |
| **1** | 1 | 1 | 0 | 0 | 0 | 0 |
| **2** | 1 | 0 | 1 | 1 | 0 | 0 |
| **3** | 0 | 1 | 1 | 0 | 1 | 0 |
| **4** | 0 | 0 | 0 | 0 | 1 | 1 |
| **5** | 0 | 0 | 0 | 1 | 0 | 1 |

Таблица 4 - Список рёбер

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **{1, 2}** | **{1, 3}** | **{1, 3}** | **{2, 5}** | **{3, 4}** | **{4, 5}** |

## Структура данных «однонаправленный список»

Списки смежности можно хранить, используя два вложенных однонаправленных списка. Однонаправленный список – структура данных, состоящая из элементов, последовательно связанных между собой с помощью указателей. Каждый элемент списка имеет указатель на следующий элемент, а последний элемент списка указывает на nil. В подобном списке можно передвигаться только в сторону его конца: узнать адрес предыдущего элемента, опираясь на содержимое текущего узла, невозможно.

## Обходы графов в глубину и ширину

Обходом графа называется процесс посещения каждой вершины графа. Результатом обхода может служить новая нумерация вершин в графе. Существуют два варианта обхода графа: в глубину и в ширину, отличающиеся порядком посещения вершин. В любом случае процесс начинается из указанной заранее вершины.

Обход в глубину использует метод поиска с возвращением, при котором после посещения очередной вершины поиск может либо продолжиться c переходом к смежной вершине, либо произойти откат к предыдущей рассмотренной вершине, если у рассматриваемой не осталось смежных с ней и не посещённых ранее вершин.

Противоположный метод обхода – обход в ширину. В этом случае у очередной вершины сначала посещаются вершины из её окрестности, после чего процесс переходит к окрестностям этих вершин.

Обходы могут описываться в терминах рекурсии, но для итеративной реализации потребуются такие структуры данных как стек и очередь. При посещении новой вершины обходом в глубину её добавляют в стек, а при необходимости отката извлекают последнюю добавленную вершину из стека. Обратный процесс происходит для поиска в ширину: для посещения очередной вершины её сначала извлекают из начала очереди, а после посещения и перед переходом к следующей вершине все вершины из её окрестности помещают в конец очередь. Обходы завершаются в том случае, когда посещены все вершины, иначе говоря, когда стек или очередь станут пусты.

Частным случаем обхода графа в глубину и ширину является обход дерева. Деревом называют граф, не содержащим циклов. Это свойство позволяет быть уверенным, что при обходе никакая из вершин не будет посещена дважды, однако в графах с циклами неизвестно заранее, была ли посещена вершина ранее или нет из-за чего обход может замедлиться или зациклиться. Чтобы избежать этого, вершины после посещения «закрашивают» в некий цвет и перед посещением следующей проверяют цвет вершины. В программе это означает, что необходимо дополнительно хранить массив флагов размером, равным количеству вершин, и изменять значения флагов при посещении очередной.

На деревьях удобно изображать процесс обхода графа. К тому же, понятно их название: при обходе в глубину процесс «спускается всё ниже», пока не дойдёт до самой отдалённой от корня, «глубокой», вершины, а при обходе в ширину процесс идёт «поярусно», посещая все вершины, одинаково удалённые от корня.

В таблице 5 номерами вершин показан порядки обхода дерева.

Таблица 5 - Обходы графа

|  |  |
| --- | --- |
| Обход в глубину | Обход в ширину |
|  |  |

Что касается временной сложности, оба алгоритма имеют одну временную сложность, равную O(n+m).

Для работы алгоритмов необходимы стек или очередь, хранящие от 0 до n вершин графов и n флагов для хранения посещённости вершин, тогда пространственная сложность алгоритмов составляет O(n).

Оба алгоритма одинаково сложны, но для разных видов графов результаты могут отличаться: в графах, где много «тупиков» произойдёт много откатов, а в графе простой цикл откаты будут происходить один за другим после того, как все вершины уже будут помечены, и в задачах поиска могут не происходить, если вершина была найдена.

Обходы в глубину и ширину используют для решения многих задач:

* проверка связности вершин;
* проверка графа на ацикличность;
* проверка графа на двудольность;
* топологическая сортировка;
* выделение связных компонент;
* поиск точек сочленения, мостов;
* и т.д.

Главное отличие алгоритмов состоит в следующем свойстве обхода в ширину: вершина в каждом «ярусе» удалена от начальной на столько же рёбер, каков номер этого «яруса». Таким образом, алгоритм поиска в ширину можно использовать для эффективного нахождения расстояний между вершинами в простых графах. При движении «в глубину» для этого приходилось бы перебирать все простые пути между двумя вершинами.

К тому же алгоритмы используются и для задач, которые можно сформулировать в терминах графов. К примеру, область применения обхода в ширину – поиск в социальных сетях, нахождение соседних узлов в одноранговых сетях вроде BitTorrent, поисковые системы для построения индексов веб-страниц. А наиболее известное применение обхода в глубину – решения головоломок с одним решением вроде лабиринта.

В графах с ориентированными рёбрами (орграфы), петлями (псевдографы) и кратными рёбрами (мультиграфы) алгоритмы работают аналогично.

## Алгоритм Дейкстры для взвешенных графов

Алгоритм был изобретён нидерландским учёным Эдсгером Дейкстрой в 1959 году и используется для поиска кратчайшего расстояния между двумя вершинами во взвешенном графе с неотрицательными весами. При этом с помощью некоторых модификаций возможно сохранение кратчайшего пути, а также нахождение всех расстояний и кратчайших путей из заданной вершины до остальных. Алгоритм можно использовать для простых графов, орграфов, псевдографов и мультиграфов.

Важными понятиями для алгоритма являются понятия временных и постоянных меток, а также процесса релаксации.

Временная метка вершины - верхняя оценка расстояния между текущей вершиной и начальной. Она вычисляется с помощью релаксаций рёбер, инцидентных с вершиной, у которой рассматривается постоянная метка.

Постоянная метка вершины - расстояние между текущей вершиной и начальной. На каждом шаге алгоритма для одной вершины выбирается постоянная метка как минимальная из временных меток.

Изначально, метка начальной вершины считается постоянной и равна 0, а для остальных вершин временным меткам присваивается максимально допустимое значение.

Релаксация по ребру (i, j) заключается в следующем: из найденной временной метки вершины j и суммы текущей постоянной метки вершины i с весом ребра (i, j) выбирается минимальное значение и сохраняется во временную метку j. Таким образом, релаксация - процесс уменьшения временных меток.

Тогда суть алгоритма заключается в следующем: на очередном шаге находится постоянная метка и для соответствующей ей вершины происходят релаксации по инцидентным данной вершине рёбрам. Если же постоянной метке соответствовала искомая вершина, то процесс завершается.

Если необходимо найти сам кратчайший путь, то для сохранения пути необходимо на каждом шаге записывать вершину, соответствующую постоянной метке, в конец уже найденного пути и по завершении цикла записать в конец пути искомую вершину.

Метод схож с поиском в ширину в том смысле, что исследуется окрестность текущей вершины, после чего вершина помечается как посещённая (постоянная метка и закрашенные вершины). При этом вместо добавления в очередь не закрашенных вершин, происходят релаксации по рёбрам, инцидентным этой вершине, а вместо извлечения вершины из начала очереди выбирается вершина с минимальной временной меткой.

Если искомая вершина самая удалённая, то происходит n выборов постоянных меток, а также происходит m релаксаций. В зависимости от используемых структур данных для хранения меток сложность релаксации и выбора постоянной метки могут разниться. В случае массива: релаксация происходит за O(1), а поиск минимума за O(n), тогда в худшем случае временная сложность алгоритма - O(n2+m). Для алгоритма могут быть применены такие древовидные структуры как двоичная и фибоначиева кучи.

Таблица 6 – Сложность алгоритма Дейкстры

|  |  |  |  |
| --- | --- | --- | --- |
| Структура данных | Поиск минимума | Запись элемента | Сложность алгоритма |
| Массив |  |  |  |
| Двоичная куча |  |  |  |
| Фибоначиева куча |  |  |  |

Алгоритм является жадным, то есть на каждом шаге принимает наиболее оптимальное решение для текущего шага, которое в дальнейшем не пересматривается.

Таким образом, после выбора постоянной пометки вершины предполагается, что любой другой путь приведёт к большей длине, чем рассмотренный на текущем шаге. Однако в графах с отрицательными весами это не всегда верно: даже одно ребро не рассмотренного пути может уменьшить расстояние так, что оно станет меньше, чем постоянная метка.

Это можно увидеть на примере орграфа с рисунка 1.2: при поиске из вершины 1 до вершины 3 постоянные метки (указаны около вершин) равны 0, 1, 2, а действительные расстояния до каждой из вершин: 0, -8, 2.

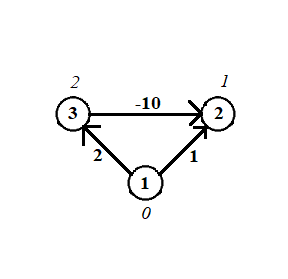


Рисунок .2 – Орграф с отрицательными весами

К тому же, в графах с отрицательными весами кратчайших путей может не существовать вовсе, так как дополнительные проходы по циклу с отрицательным суммарным весом уменьшают кратчайший путь, что верно для графа на рисунке 1.3.

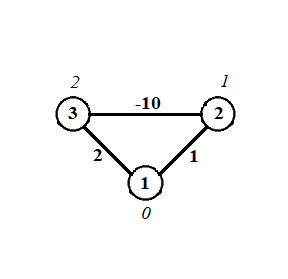


Рисунок 1.3 - Простой цикл с отрицательными весами

Таким образом, для поиска кратчайших путей в графах с отрицательными весами используют другие алгоритмы, к примеру, алгоритмы Беллмана-Форда и Флойда-Уоршелла.

Поиск кратчайшего пути может использоваться для построения маршрута в приложениях вроде Google Maps, при создании компьютерных сетей для обеспечения минимальной задержки, в абстрактных автоматах для определения оптимальной стратегии и т.д.

## Работа с текстовыми файлами

Файловый тип – произвольная последовательность элементов, длина которой заранее не определена, а конкретизируется в процессе выполнения программы. Это определение логического файла, т.е. того, который используется в программе. Физический файл – это поименованная область памяти на внешнем носителе, в которой хранится некоторая информация.

Всего существует три типа файлов:

* типизированные файлы;
* текстовые файлы;
* нетипизированные файлы.

Типизированные файлы связываются с файловыми переменными, объявленными как file of <Тип>. Файл считается состоящим из элементов, каждый из которых имеет тип <Тип>.

Нетипизированные файлы могут быть связаны только с теми файловыми переменными, которые были объявлены как file. Файл считается состоящим из элементов, размер которых определяется при открытии файла.

Текстовый файл представляет собой последовательность символов, однако он не эквивалентен файлу типа file of Char. Текстовые файлы связываются с файловыми переменными, принадлежащими стандартному типу TextFile. Особенность текстовых файлов состоит в том, что содержащиеся в них символы разбиваются на строки. Строки имеют различную длину, а в конце каждой помещается специальный управляющий символ: возврат каретки и символ перехода на новую строку.

Для доступа к отдельным элементам в Delphi существуют специальные стандартные процедуры и функции. Их называют процедурами и функциями ввода-вывода.

Процедура AssignFile (F, Name) связывает файловую переменную с внешним файлом на диске. Здесь F – имя файловой переменной любого типа; Name – выражение строкового типа. Назначение процедуры: организует связь между конкретным физическим файлом на внешнем устройстве (конкретным набором данных) и файловой переменной (логическим файлом).

Процедура Reset (F) открывает существующий файл F. При этом открывается внешний файл с именем, присвоенным переменной F процедурой AssignFile. Если файла не существует, возникает сообщение об ошибке. Указатель файла устанавливается на первую позицию файла. Если файл был предварительно открыт, то он закрывается и повторно открывается. При выполнении процедуры содержимое файла не изменяется. Для текстовых файлов файл открывается в режиме только для чтения (read-only).

Процедура Rewrite (F) создает новый файл и открывает его. Если файл уже открыт, то закрывает и снова открывает его. Указатель файла устанавливается на начало файла (файл создается пустым). Для текстовых файлов – в режиме только для записи (write-only).

Процедура Append (F) предопределена только для текстовых файлов. Она открывает существующий файл для добавления. Если файл уже открыт, то закрывает его, а затем вновь открывает. В данном случае указатель файла устанавливается на конец файла. Файл открывается в режиме только для записи.

Для закрытия файла используется процедура CloseFile (F), где F – файловая переменная, открытая с помощью Reset, Rewrite или Append.

Функции Eof (F) и Eol(F) проверяют на конец файла или символ конца строки соответственно.

Для файлов с типом предопределены процедуры считывания компонентов файла в переменные и записи в файл компонентов из переменных Read (F, V1, V2, …, Vn) и Write (F, V1, V2, …, Vn).

Для файлов без типа есть аналоги процедур Read и Write. Это процедуры BlockRead (F, Buf, Count [, Done]) и BlockWrite (F, Buf, Count [, Done]).

Также есть функции и процедуры, перемещающие указатель файла в указанную позицию, возвращающие текущее положение указателя файла в байтах, возвращающие размер файла в компонентах и др.

Экспортировать полученные графы можно в векторном формате, который основан на математическом описании геометрических объектов. Для этой цели был выбран формат SVG, являющийся стандартизированным расширением языка разметки XML.

SVG-файлы содержат XML-разметку, описывающую векторные изображения: все детали изображения в виде геометрических примитивов (линий, кругов, прямоугольников и т. д.) с указанием размеров, координат и другой необходимой информации.

XML-файлы имеют древовидную структуру: в файле имеется корневой узел, имеющий потомков. Узлы представляются в виде тегов – текста, записанного в угловых скобках. Помимо корневого элемента не вложенными могут быть некоторые теги, необходимые для объявления типа документа и другой служебной информации, а именно: <?xml?> и <!DOCTYPE>.

Теги могут быть одинарными и парными. Первые должны иметь перед закрывающей скобкой символ слеш (/). У вторых есть открывающий и закрывающий теги с одинаковыми именами. У закрывающего тега прямо после открывающей скобки должен стоять слеш. Между открывающим и закрывающим тегом могут находиться текст и другие теги.

SVG-файл должен иметь корневой элемент <SVG>. Для рисования графов будут использованы следующие одинарные теги: circle, описывающий круглые геометрические фигуры, и line, описывающий прямые линии.

Для предоставления дополнительной информации у тегов могут иметься атрибуты – имена, которым могут присвоены значения через символ равно (=), которые должны находиться в апострофах. Атрибуты парных тегов записываются в открывающем теге.

Для описания свойств объектов векторного изображения в SVG-тегах применяются различные атрибуты. В частности, Fill и Stroke определяют соответственно цвета заливки и обводки фигуры.

Атрибуты для тега circle: cx, cy – координаты центра окружности, r – радиус окружности. Для тега line: x1, y1 – координаты первого конца отрезка, x2, y2 – координаты второго конца отрезка.

Для работы с XML-файлами в Delphi можно использовать компоненту TXMLDocument. За возможность чтения или записи из файла отвечает её поле Active, которое перед записью или чтением необходимо установить в значение True, а после окончания работы с файлом – в False.

Для открытия и сохранения файла могут быть использованы методы-процедуры LoadFromFile(AFileName) и SaveToFile(AFileName) соответственно, где AFileName – путь к файлу.

Для создания корневого элемента используется методы AddChild(TagName) или AddChild(TagName, NamespaceURI), где TagName –строка с именем тега (для SVG-файлов – svg), а NamespaceURI – строка с пространством имён, содержащего описание узла. Они возвращают корневой узел типа IXMLNode, имеющий аналогичные методы для создания вложенных в него тегов. Методы создают одинарные теги, если узлы не содержат потомков или текста внутри.

Для обращения к атрибутам узлов используется поле Attributes[AttrName], где AttrName – строковый тип, в котором указано имя атрибута.

# Постановка задачи

Реализовать программное средство для создания простых и ориентированных графов со взвешенными рёбрами положительного веса и демонстрации работы обхода в ширину, обхода в глубину и алгоритма Дейкстры.

Использовать типизированный файл для чтения/записи графа.

Для хранения введённых графов использовать представление графа списками смежности, представляющие собой однонаправленный список, элементы которого являются однонаправленными списками.

Доступный функционал программы:

* добавление и удаление вершины;
* добавление и удаление ребра;
* загрузка и сохранение графа из типизированного файла;
* экспорт графа в формате SVG;
* обход графа в глубину, начиная с указанной вершины;
* обход графа в ширину, начиная с указанной вершины;
* нахождение кратчайших путей и расстояний между вершинами алгоритмом Дейкстры.

# Описание алгоритмов решения задачи

Таблица 1 – Описание алгоритмов решения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Наименование алгоритма | Назначение алгоритма | Формальные параметры | Предлагаемый тип реализации |
| 1 | Инициализация главной формы |  |  |  |
| 2 | InitializeGraph |  |  |  |
| 3 | DestroyGraph |  |  |  |
| 4 | AddVertice |  |  |  |
| 5 | AddArc |  |  |  |
| 6 | DeleteVertice |  |  |  |
| 7 | DeleteArc |  |  |  |
| 8 | GetByNumber |  |  |  |
| 9 | OpenGraph |  |  |  |
| 10 | SaveGraph |  |  |  |
| 11 | ImportGraph |  |  |  |
| 12 | ExportGraph |  |  |  |
| 13 | ToWeightMatrix |  |  |  |
| 14 | DFS |  |  |  |
| 15 | BFS |  |  |  |
| 16 | Dijkstra |  |  |  |
| 17 | MinDist |  |  |  |
| 18 | InitializeList |  |  |  |
| 19 | DestroyList |  |  |  |
| 20 | Push |  |  |  |
| 21 | Pop |  |  |  |
| 22 | Queue |  |  |  |
| 23 | Dequeue |  |  |  |
| 24 | Visit |  |  |  |
| 25 | MakePassive |  |  |  |
| 26 | MakeRegularPolygon |  |  |  |

# Структура данных

## Структура типов программы

Таблица 2 – Структура типов программы

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| TDesign |  |  |
| TPNeighbour |  |  |
| TNeighbour |  |  |
| TPVertice |  |  |
| TVertice |  |  |
| TGraph |  |  |
| TWeights |  |  |
| TVerFile |  |  |
| TArcFile |  |  |
| TPItem |  |  |
| TItem |  |  |
| TStack |  |  |
| TQueue |  |  |