Оглавление

Введение

Была разработана программа для визуализации и работы с графами под названием «Ultimate Graph». Программа имеет множество функций для работы с графами, таких как создание графов, редактирование. Главной особенностью программы является визуализация алгоритмов. При включении алгоритма, пользователь задает таймер времени, то есть промежуток времени, за который алгоритм будет совершать одно действие, после чего данный алгоритм визуализируется и в отдельном окне пишется, почему и как происходит данное действие.

Постановка задачи

Основными задачами, которые призвана решать разработанная программа, являются следующие:

- Подробное описание и визуализация алгоритмов на графах.
- Вычисление различных характеристик графов.

Обзор аналогов

Существует множество уже написанных программ для работы с графами:

- newGraph
- maple
- mathematica
- quickGraph
- набор визуализаторов на сайте Санкт-Петербургского Государственного университета информационных технологий, механики и оптики

Однако разработанная программа имеет ряд преимуществ перед аналогами. Данный продукт создан для максимального разъяснения, визуализации и понимания теории графов. В ней присутствует развитый легко настраиваемый графический интерфейс, способствующий быстрому ознакомлению и эффективному использованию всех возможностей программы.

Средства выполнения проекта

Языки программирования и технологии визуализации

Средой программирования была избрана система Microsoft Visual C# 2005 из-за сочетания таких факторов, как удобный интерфейс, мощный редактор кода, хороший отладчик и оптимизирующий компилятор, параметры которой легко настроить под конкретный проект.

Для оформления отдельных деталей интерфейса была задействована одна из самых популярных и доступных систем редактирования графических изображений — Adobe Photoshop CS 3. С помощью нее были выполнены некоторые элементы дизайна.

Графы

Основные понятия

В математической теории графов и информатике граф — это совокупность объектов со связями между ними. Теория графов является частью дискретной математики.

Объекты представляются как вершины, или узлы графа, а связи — как дуги, или рёбра. Для разных областей применения виды графов могут различаться направленностью, ограничениями на количество связей и дополнительными данными о вершинах или рёбрах.

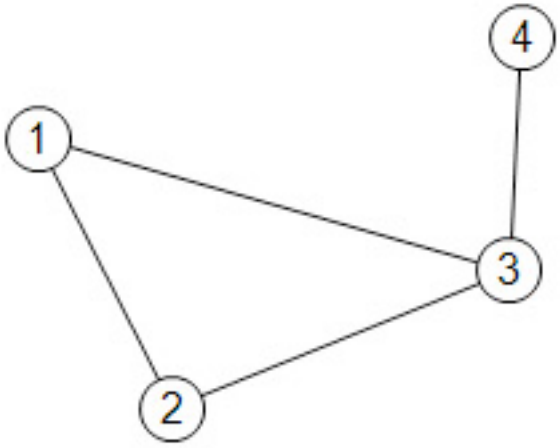

Рис. 1. Пример связного невзвешенного и неориентированного графа.

Графами могут быть представлены блок-схемы программ для ЭВМ, сетевые графики строительства, где вершины — события, означающие окончания работ на некотором участке, а рёбра, связывающие эти вершины, — работы, которые возможно начать по завершению одного события и необходимо выполнить для совершения следующего.

Многие структуры, представляющие практический интерес в математике и информатике, могут быть представлены графами. Например, строение энциклопедии можно смоделировать при помощи ориентированного графа, в котором вершины — это статьи, а дуги — это перекрестные ссылки между статьями.

Немного теории

Неориентированный граф G — это упорядоченная пара $G := (V,E)$, для которой:

- V — это множество вершин или узлов.
- E — это множество пар (в случае неориентированного графа — неупорядоченных) вершин, называемых рёбрами.

Вершины и рёбра графа называются также элементами графа, число вершин в графе $|V|$ — порядком графа, число рёбер $|E|$ — размером графа.

Основные определения:

- Вершины u и v называются *концевыми вершинами* (или просто концами) ребра $e = \{u,v\}$. Говорят, что ребро, в свою очередь, соединяет эти вершины. Две концевые вершины одного и того же ребра называются соседними, или смежными.
- Два ребра называются *смежными*, если они имеют общую концевую вершину.
- Два ребра называются *кратными*, если множества их концевых вершин совпадают.
- Ребро называется *петлёй*, если его концы совпадают, то есть $e = \{v,v\}$.
- Вершина называется *изолированной*, если она не является концом ни для одного ребра; висячей (или листом), если она является концом ровно одного ребра.
- *Связным* графом называется граф, содержащий ровно одну компоненту связности. Это означает, что между любой парой вершин этого графа существует, по крайней мере, один путь.
- *Взвешенный* графом называют граф, некоторым элементам которого (вершинам, рёбрам или дугам) сопоставлены числа. Их обычно называют весом, длинной или стоимостью.
- *Остовным* подграфом называют подграф, множество вершин которого совпадает с множеством вершин исходного графа.
- *Остовом* (неориентированного) связного графа G=(V,E) называется его частичный граф S=(V,T), являющийся деревом.

- *Деревом* называют связный граф, не содержащий циклов.
- *Бесконтурным ориентированным графом* (или направленным ациклическим графом) называют случай направленного графа, в котором отсутствуют направленные циклы, то есть пути, начинающиеся и кончающиеся в одной и той же вершине. Направленный ациклический граф является обобщением дерева.
- *Частично упорядоченное множество* называют математическое понятие, которое формализует интуитивные идеи упорядочивания, расположения в определенной последовательности и т. п. Неформально говоря, множество частично упорядочено, если указано, какие элементы следуют (больше и т. п.) за какими.
- *Топологической сортировкой* называют упорядочивание вершин бесконтурного ориентированного графа согласно частичному порядку, заданному рёбрами орграфа на множестве его вершин.

Применение теории графов

- В химии (для описания структур, путей сложных реакций, правило фаз также может быть интерпретировано как задача теории графов). Теория графов позволяет точно определить число теоретически возможных изомеров у углеводородов и других органических соединений.
- В информатике и программировании.
- В коммуникационных и транспортных системах. В частности, для моделирования сетей.
- В экономике (некоторые методы в теории графов позволяют решать производственно-экономические задачи).
- В схемотехнике.

Алгоритмы на графах

На графах существует огромное множество алгоритмов, в данной программе приведены некоторые классические, наиболее известные алгоритмы.

Поиск в ширину

Поиск в ширину (англ. *Breadth-first search, BFS*) — метод обхода и разметки вершин графа.

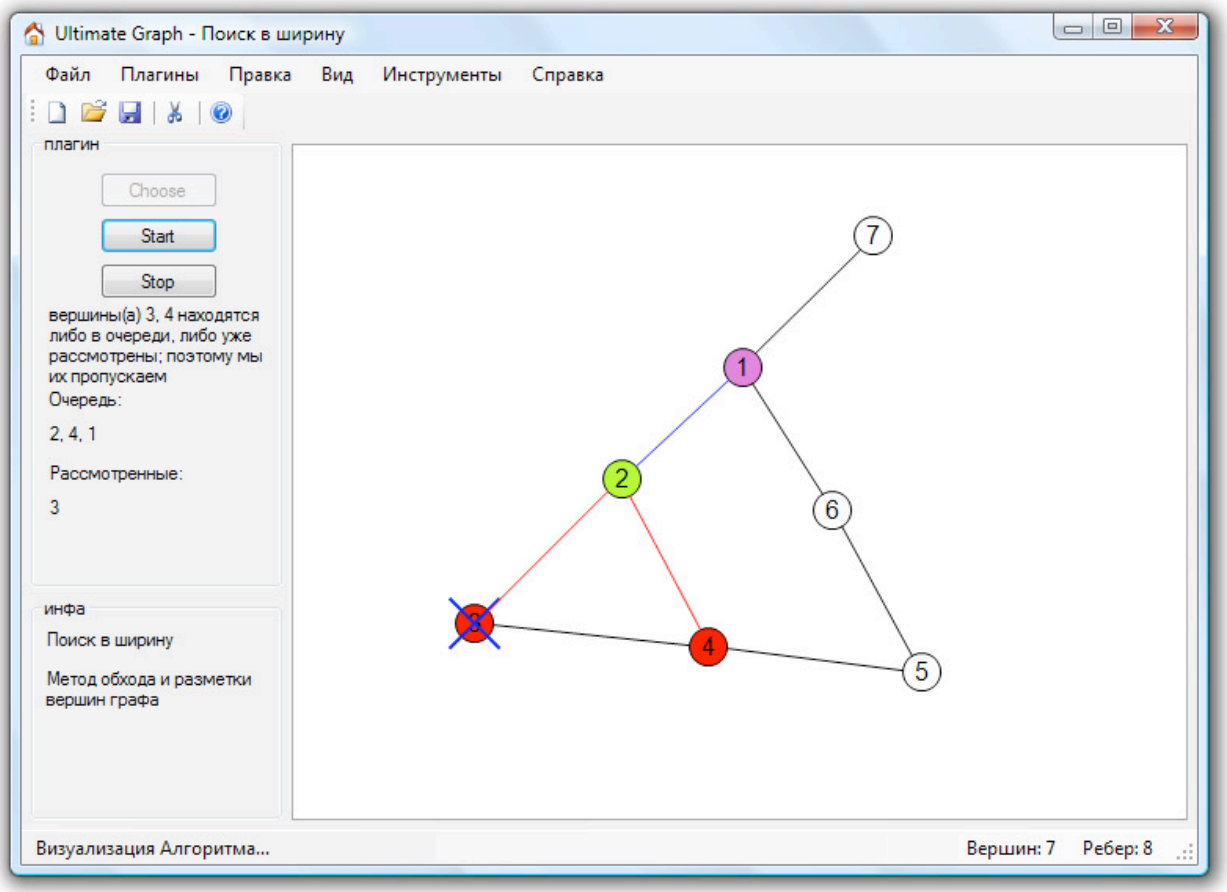

Рис. 2. Визуализация алгоритма поиска в ширину.

Поиск в ширину выполняется в следующем порядке: началу обхода s приписывается метка 0, смежным вершинам — метка 1. Затем поочередно рассматривается окружение всех вершин с метками 1, и каждой из входящих в эти окружения вершин приписываем метку 2 и т. д.

Если исходный граф связный, то поиск в ширину пометит все его вершины. Дуги вида (i, i+1) порождают остовное дерево.

В результате поиска в ширину находится путь кратчайшей длины в невзвешененном графе, т.е. путь, содержащий наименьшее число рёбер. Алгоритм работает за $O(n+m)$, где n — количество вершин, m — рёбер.

Применения алгоритма:

- Поиск кратчайшего пути в невзвешенном графе.
- Нахождения решения какой-либо задачи с наименьшим числом ходов, если каждое состояние системы можно представить вершиной графа, а переходы из одного состояния в другое — рёбрами графа.
- Нахождение кратчайшего пути.
- Нахождение кратчайшего цикла в неориентированном невзвешенном графе.
- Нахождение всех ребер, лежащих на кратчайшем пути между заданной парой вершин.
- Нахождение всех ребер, лежащих на всех кратчайших путях.
- Нахождение кратчайшего чётного пути в графе (т.е. путь чётной длины).
- Нахождение всех вершин, лежащих на кратчайшем пути.

Поиск в глубину

Поиск в глубину (англ. *Depth-first search, DFS*) — один из методов обхода графа. Алгоритм поиска описывается следующим образом: для каждой непройденной вершины необходимо найти все непройденные смежные вершины и рекурсивно повторить поиск для них. Используется в качестве подпрограммы в алгоритмах поиска одно- и двусвязных компонент, топологической сортировки.

В результате поиска в глубину находится лексикографически первый путь в графе. Алгоритм работает за $O(n+m)$.

Применения алгоритма:

- Проверка, является ли одна вершина дерева предком другой.
- Топологическая сортировка.
- Проверка графа на ацикличность и нахождение цикла.
- Поиск компонент сильной связности.
- Поиск мостов.

Поиск мостов

Пусть дан связный неориентированный граф. Мостом называется такое ребро, удаление которого делаем граф несвязным. Алгоритм работает за $O(n+m)$.

Алгоритм

Запустим обход в глубину из произвольной вершины графа. Заметим следующий факт.

Если текущее ребро таково, что ведёт в вершину, из которой и из любого её потомка нет обратного ребра в текущую вершину или её предка, то это ребро является мостом. В противном случае оно мостом не является.

Теперь осталось научиться для каждой вершины эффективно проверять, не найдётся ли из её потомка обратное ребро в текущую вершину или её предка. Для этого воспользуемся временами входа поиска в глубину.

Итак, пусть $tin[v]$ — это время захода поиска в глубину в вершину v. Теперь введём массив $fup[v]$, который и позволит нам отвечать на вышеописанные запросы. Время $fup[v]$ равно минимуму из времени захода в саму вершину $tin[v]$, времён захода в каждую вершину p, являющуюся концом некоторого обратного ребра (v,p), а также из всех значений $fup(to)$для каждой вершины to, являющейся непосредственным сыном v в дереве поиска:

$$fup[v] = \min \begin{cases} tin[v] \\ tin[p], \text{для всех } (v, p) \text{ — обратное ребро} \\ fup[to], \text{для всех } (v, to) \text{ — ребро дерева} \end{cases}$$

Тогда, из вершины v или её потомка есть обратное ребро в её предка тогда и только тогда, когда найдётся такой сын to, что $fup[to] < tin[v]$. Если $fup[to] = tin[v]$, то это означает, что найдётся обратное ребро, приходящее точно в v.

Таким образом, если для текущего ребра (v, to) (принадлежащего дереву поиска) выполняется $fup[to] > tin[v]$, то это ребро является мостом; в противном случае оно мостом не является.

Алгоритм Дейкстры

Алгоритм Дейкстры находит кратчайшее расстояние от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса. Алгоритм широко применяется в программировании и информационных технологиях, например, его использует протокол OSPF для устранения кольцевых маршрутов. Известен также под названием «кратчайший путь — первый» (англ. *Shortest Path First, SPF*).

Постановка задачи

Дан ориентированный или неориентированный взвешенный граф с n вершинами и m рёбрами. Веса всех рёбер неотрицательны. Указана некоторая стартовая вершина s. Требуется найти длины кратчайших путей из вершины s во все остальные вершины, а также предоставить способ вывода самих кратчайших путей.

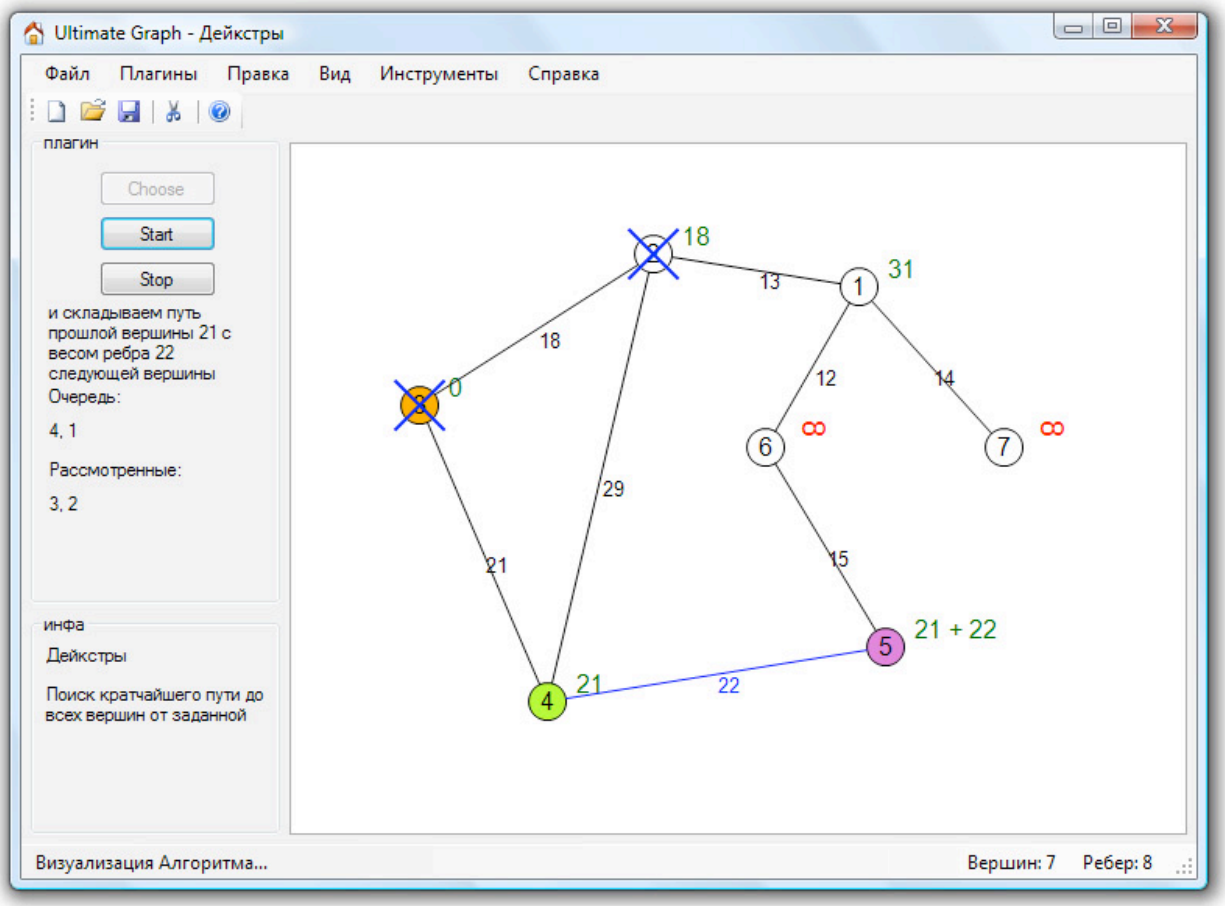

Рис. 3. Визуализация алгоритма Дейкстры.

Алгоритм

Заведём массив $d[]$, в котором для каждой вершины v будем хранить текущую длину $d[v]$ кратчайшего пути из s в v. Изначально $d[s] = 0$, а для всех остальных вершин эта длина равна бесконечности (при реализации на компьютере обычно в качестве бесконечности выбирают просто достаточно большое число, заведомо большее возможной длины пути):

$$d[v] = \infty, v \neq s$$

Кроме того, для каждой вершины v будем хранить, помечена она ещё или нет, для этого заведем массив $u[]$. Изначально все вершины не помечены, т.е. $u[] = \text{false}$.

Сам алгоритм Дейкстры состоит из n итераций (повторений). На очередной итерации выбирается вершина v с наименьшей величиной $d[v]$ среди ещё не помеченных, т.е.:

$$d[v] = \min_{p : u[p] = \text{false}} d[p]$$

На первой итерации выбрана будет стартовая вершина s.

Выбранная таким образом вершина v отмечается помеченной. Далее, на текущей итерации, из вершины v производятся релаксации: просматриваются все рёбра (v, to), исходящие из вершины v, и для каждой такой вершины to алгоритм пытается улучшить значение $d[to]$. Пусть длина текущего ребра равна len, тогда в виде кода релаксация выглядит как:

$$d[to] = \min(d[to], d[v] + \text{len})$$

На этом текущая итерация заканчивается, алгоритм переходит к следующей итерации (снова выбирается вершина с наименьшей величиной d, из неё производятся релаксации, и т.д.). При этом в конце концов, после n итераций, все вершины графа станут помеченными, и алгоритм свою работу завершает. Утверждается, что найденные значения $d[v]$ и есть искомые длины кратчайших путей из s в v.

Стоит заметить, что, если не все вершины графа достижимы из вершины s, то значения $d[v]$ для них так и останутся бесконечными. Понятно, что несколько последних итераций алгоритма будут как раз выбирать эти вершины, но никакой полезной работы производить эти итерации не будут (поскольку бесконечное расстояние не сможет прорелаксировать другие, даже тоже бесконечные расстояния). Поэтому алгоритм можно сразу останавливать, как только в качестве выбранной вершины берётся вершина с бесконечным расстоянием.

Разумеется, обычно нужно знать не только длины кратчайших путей, но и получить сами пути. Покажем, как сохранить информацию, достаточную для последующего

восстановления кратчайшего пути из s до любой вершины. Для этого достаточно так называемого массива предков: массива $p[]$, в котором для каждой вершины $v \neq s$ хранится номер вершины $p[v]$, являющейся предпоследней в кратчайшем пути до вершины v. Здесь используется тот факт, что если мы возьмём кратчайший путь до какой-то вершины v, а затем удалим из этого пути последнюю вершину, то получится путь, оканчивающийся некоторой вершиной $p[v]$, и этот путь будет кратчайшим для вершины $p[v]$. Итак, если мы будем обладать этим массивом предков, то кратчайший путь можно будет восстановить по нему, просто каждый раз беря предка от текущей вершины, пока мы не придём в стартовую вершину s — так мы получим искомый кратчайший путь, но записанный в обратном порядке. Итак, кратчайший путь P до вершины v равен:

$$P = (s,\ldots, p[p[p[v]]], p[p[v]], p[v], v)$$

Осталось понять, как строить этот массив предков. При каждой успешной релаксации, т.е. когда из выбранной вершины v происходит улучшение расстояния до некоторой вершины to, мы записываем, что предком вершины to является вершина v:

$$p[to] = v$$

Написание собственных плагинов

Что такое плагины

Плагины – это модули, динамически подключаемые к программе. Примером программного продукта, активно использующего технологию плагинов, является браузер Mozilla Firefox, где даже пользовательский интерфейс представляет собой подключаемый модуль, допускающий практически произвольную модификацию. Примером плагина является библиотека классов с расширением dll или специальным форматом, написанная разработчиками программы.

Как создавать плагины

Плагины в Ultimate Graph пишутся на языке C#.

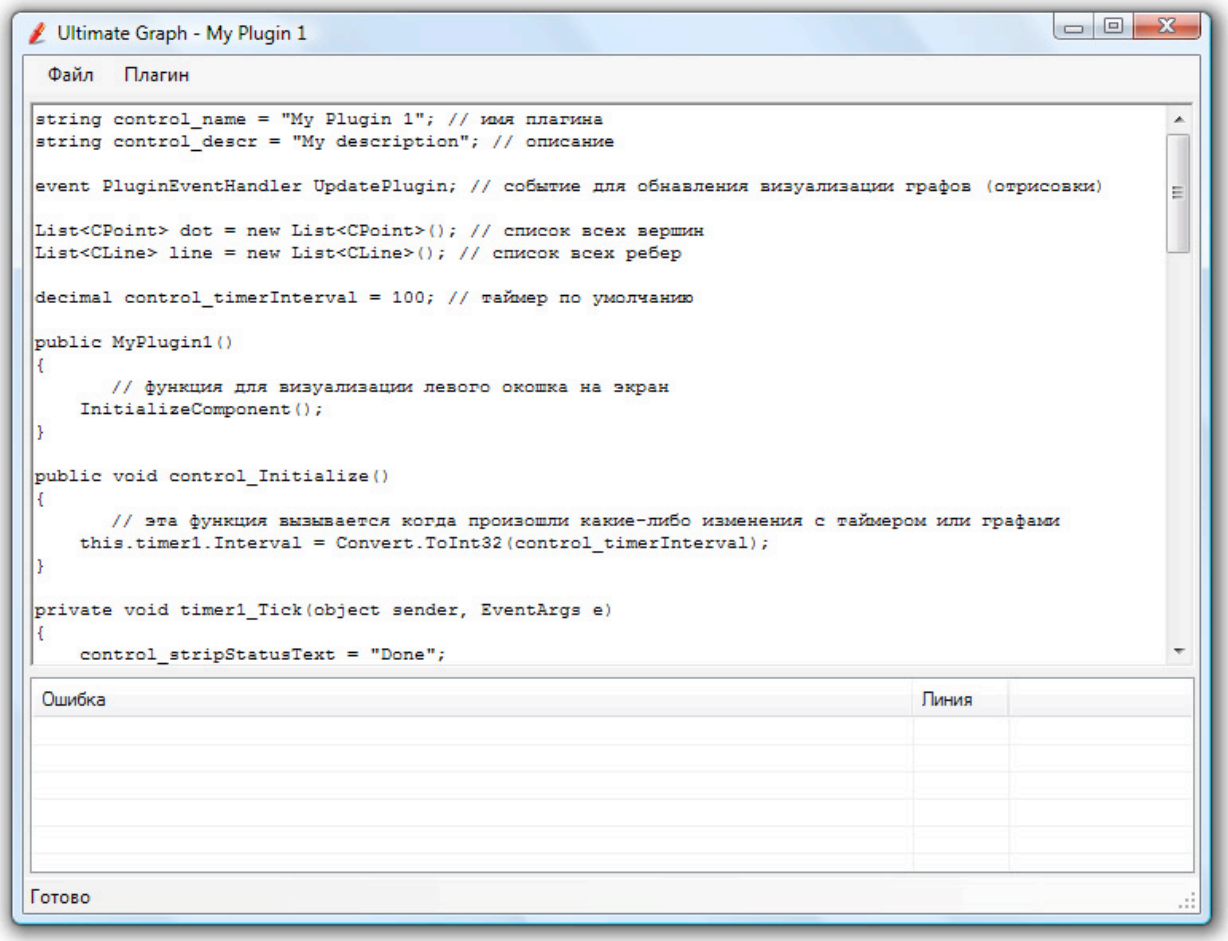

Рис. 4. Главное окно редактора плагинов.

Для того чтобы вызвать редактор плагинов, в главном окне программы выберем Файл » Редактор Плагинов (Ctrl + P).

Далее выберите Файл » Новый, чтобы создать новый плагин. Перед вами появится куски кода. Это уже готовый шаблон, благодаря которому легче писать алгоритмы.

По умолчанию создалась функция timer1_Tick, в которой написан основной алгоритм. После того, когда проходит некий промежуток времени (задающийся пользователем), программа снова вызывают функцию timer1_Tick, поэтому пользователь должен обустроить эту функцию так, чтобы в ней не происходило одно и то же действие. Для этого можно просто менять одну и ту же переменную, и в случаи, когда нам нужно вернуться назад, мы просто присваиваем ей начальное значение. Когда алгоритм нужно завершить, пользователь его останавливает функцией timer1.Stop().

Главные переменные:

- dot – список всех вершин (задается пользователем)
  - mx – координата по оси oX
  - my – координата по оси oY
  - vertex – номер вершины
  - color – цвет вершины (по умолчанию Black)
  - sup – верхний индекс
  - sup_color – цвет верхнего индекса (по умолчанию DarkCyan)
  - cross – крест вдоль всей вершины
  - cross_color – цвет креста (по умолчанию Blue)
  - highlight_color – задний фон вершины (по умолчанию LightBlue)
- line – список всех ребер (задается пользователем)
  - lx_1 – начальная координата ребра по oX
  - ly_1 – начальная координата ребра по oY
  - lx_2 – конечная координата ребра по oX
  - ly_2 – конечная координата ребра по oY
  - first – начальная вершина ребра
  - second – конечная вершина ребра

    - color – цвет ребра
- control_name – имя данного плагина
- control_descr – описание данного плагина
- control_timerInterval – интервал таймера в миллисекундах
- control_radius – радиус вершины
- control_cursor - курсор
- control_stripStatusText – текст в строке состояния
- control_CancelNewDot – отмена добавления новых вершин
- control_CancelNewEdge – отмена добавления новых ребер
- control_EdgeSize – функция для нахождения длины ребра
- UpdatePlugin – делегат, благодаря которому происходит обновление плагина. Это означает, что при вызыве этого делегата все изменения с графами сразу входят в работу программы, например изменение цвета вершин графа.

Заключение

Выводы

Подведём итоги. Была реализована программа для работы с графами, позволяющая строить графы и визуализировать различные алгоритмы, подстраивая их под себя.

Был подробно описан каждый алгоритм в программе, о том, как он строится и где применяется.

Направление дальнейших разработок

Основной задачей дальнейшей разработки является написание новых, более сложных алгоритмов. Также планируется дальнейшее улучшение пользовательского интерфейса и реализация новых возможностей работы с графами.

Использованные информационные источники

1. В. Липский «Комбинаторика для программистов» М.: Мир, 1988.
2. Описания различных алгоритмов:
    a. Свободная энциклопедия «ВикипедиЯ» — http://ru.wikipedia.org/wiki/Список_алгоритмов
    b. MAXimal — http://e-maxx.ru/algo
3. Официальные сайты перечня аналогов:
    a. newGraph — http://mi.sanu.ac.yu/newgraph
    b. maple — http://www.maplesoft.com
    c. mathematica — http://www.wolfram.com
    d. quickGraph — http://quickgraph.codeplex.com
    e. набор визуализаторов на сайте Санкт-Петербургского Государственного университета информационных технологий, механики и оптики — http://rain.ifmo.ru/cat/view.php/vis