1. Уточнение постановки задачи

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.

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

1.1 Алгоритмы в программе

Поиск в ширину

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

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.

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

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 $O(n+m)$.

Поиск мостов

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

Запустим обход в глубину из произвольной вершины графа. 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

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

Запустим обход в глубину из произвольной вершины графа. 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.

Алгоритм Дейкстры находит кратчайшее расстояние от одной из вершин графа до всех остальных. Известен также под названием «кратчайший путь — первый» (англ. *Shortest Path First, SPF*).

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.

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.

1.2 Написание собственных плагинов

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.

Плагины в Ultimate Graph пишутся на языке C#. 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.

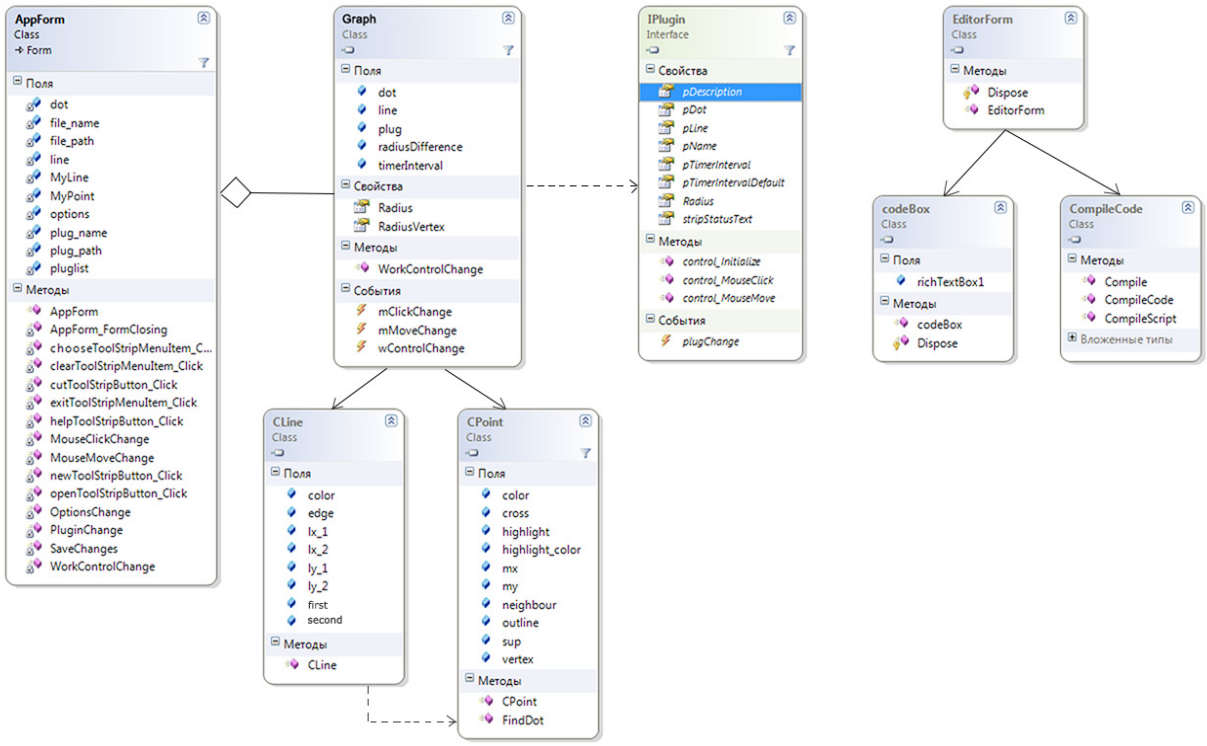
1.3 Пример плагина

Напишем собственный плагин, в котором все вершины графа закрашиваются случайным цветом по очереди. Для этого объявим в начале переменную int i=0; и изменим функцию timer1_Tick():

```
int i = 0;
private void timer1_Tick(object sender, EventArgs e) {
    if (i != dot.Count)
    {
        Random rd = new Random();
        dot[i].highlight = true;
        dot[i].highlight_color = Color.FromArgb(rd.Next(255), rd.Next(255),
rd.Next(255));
        this.label1.Text = "зарисовываю " + (i + 1) + " вершину";
        i++;
    }
    else
    {
        control_stripStatusText = "Готово";
        this.label1.Text = "Готово";
        timer1.Stop(); // останавливаем таймер
    }

    UpdatePlugin(this, e); // обновляем плагин
}
```

После компиляции в программе можно будет выбрать написанный плагин и посмотреть его работу.

2. Диаграмма классов



3. Текстовые спецификации интерфейса

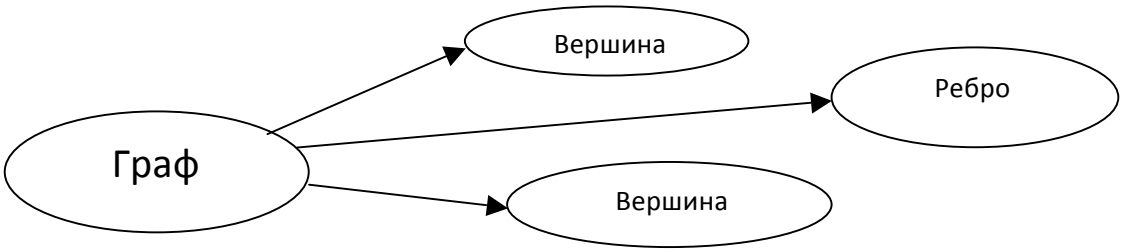
```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Data;
using System.Text;
using System.Windows.Forms;

namespace control
{
    public delegate void GraphEventHandler(object sender, EventArgs e);
    public delegate void MouseClickEventHandler(object sender, MouseEventArgs e);
    public delegate void MouseMoveEventHandler(object sender, MouseEventArgs e);
    public partial class Graph : UserControl
    {
        #region Fields
```

```
        // события любых изменений в панеле отрисовки графа
        public event GraphEventHandler wControlChange;
        public event MouseClickEventHandler mClickChange;
        public event MouseMoveEventHandler mMoveChange;

        public List<CPoint> dot;
        public List<CLine> line;

        CPoint MyPoint; // класс вершин
        CLine MyLine; // класс ребер

        ArrayList index = new ArrayList();


        // flag[0] - dots movements (drag and drop)
        // flag[1] - when we want to build the line between dots

        int[] flag = new int[2];


        // настройки отображения графов
        // checkbox1 - vertex
        // checkbox2 - arrow
        // checkbox3 - edge
        // checkbox4 - transparence
        // checkbox5 – coordinates

        public bool checkbox1 = true;
        public bool checkbox2 = false;
        public bool checkbox3 = false;
        public bool checkbox4 = true;
        public bool checkbox5 = false;

        // интервал выполнения алгоритма
        public decimal timerInterval = 1500;

        #endregion

        #region control

        public Graph()
        {
            InitializeComponent();
        }

        // классы работы с событиями мышки
        public void GraphChange(object sender, EventArgs e, int what) { }
        private void Graph_Paint(object sender, PaintEventArgs e) {}
        private void Graph_MouseMove(object sender, MouseEventArgs e) {}
        private void Graph_MouseDown(object sender, MouseEventArgs e) {}
        private void Graph_MouseDoubleClick(object sender, MouseEventArgs e) {}
        private void Graph_MouseUp(object sender, MouseEventArgs e) {}
    }
}
```

```
//-------------------------------------------------------------------------------------------------------------+


using System;
using System.Drawing;
using System.Collections.Generic;
using System.Text;

namespace control
{
    #region CPoint

    public class CPoint
    {
        // аттрибуты вершин (цвет, номер, закрашенность…)

        public Color color = Color.Black;
        public Color cross_color = Color.Blue;
        public Color sup_color = Color.DarkGreen;
        public Color highlight_color = Color.Orange;
        public Color cord_color = Color.Gray;
        public Color vertex_color = Color.Black;
        public Color outline = Color.Transparent;

        public string sup;
        public int mx, my, sx, sy, vertex;
        public bool cross, highlight;

        // функции отрисовки вершин и их аттрибутов

        internal void DrawDot(Graphics gR, int r) {}
        internal void DrawNonTransparency(Graphics gR, int r) {}
        internal void DrawCoordinates(Graphics gR, int r) {}
        internal void DrawCross(Graphics gR, int r) {}
        internal void DrawHighLight(Graphics gR, int r) {}
        internal void DrawOutline(Graphics gR, int r) {}
        internal void DrawSup(Graphics gR, int r) {}
        internal void DrawDotVertex(Graphics gR, int r) {}

    #endregion

    #region CLine

    public class CLine
    {
        // аттрибуты ребер (цвет, ширина, вес)

        public static Color temp_color = Color.Black;
        public Color color = Color.Black;
        public int width = 1;
        public int first, second, lx_1, ly_1, lx_2, ly_2, edge;


        // функции отрисовки ребер и их аттрибутов

        internal void DrawRoundDot(int r) {}
        internal void DrawLine(Graphics gR) {}
```

```
        internal void DrawArrow(Graphics gR) {}
        internal void DrawEdge(Graphics gR) {}

    #endregion
}
```

4. Диаграмма объектов



5. Инструментальные средства

Средой программирования была избрана система Microsoft Visual C# 2010 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.

6. Описание файловой структуры системы

AppForm.cs – основное окно программы
control.cs – панель для работы с классом Graph
Plugins.cs – окно, в котором выводится список плагинов
About.cs – окно “О Программе”
pluginfo.cs – интерфейс для динамического подключения плагинов к программе
EditorForm.cs – окно для создания своих плагинов
Compiler.cs – класс компилятора
controlCPoint.cs – классы вершин и ребер
breadthSearch.dll – алгоритм поиска в ширину
depthSearch.dll – алгоритм поиска в глубину
dijkstra.dll – алгоритм Дейкстры
bridges.dll – алгоритм поиска мостов

7. Пользовательский интерфейс

При запуске программы, пользователь при помощи левым нажатием мыши в области рисования может создавать новые вершины и ребра. Двойным щелчком правой мыши вершина и соответствующие к ней ребра удаляются. Если зажать левую кнопку мыши, то вершина перемещается. В меню Настройки -> Инструменты можно задать вид отображения графов, добавить веса и ориентацию. В меню Плагины -> Выбрать список плагинов.