Отчет

1. Проведен обзор алгоритмов и подходов к решению задачи раскладки графовых структур в двумерном пространстве, рассмотрены особенности force-directed алгоритмов.
2. Запрограммирован визуализатор, чьи функции включают отображение графовой структуры на плоскости и сохранение получившегося результата в файле картинки
3. Реализован алгоритм поиска соседних вершин для заданной, основанный на принципе разбиения всех имеющихся на плоскости вершин по так называемым блокам. За данную функциональность отвечает метод:

public void CreateBlocks(IList<Coordinate> coordinate, System.Drawing.Size size);

который принимает на вход коллекцию вершин и размеры плоскости, где будет происходить отрисовка. Метод создает массив из объектов класса Coordinate, каждый элемент массива будет содержать координаты левого верхнего угла каждого блока. Число блоков вычисляется исходя из максимального радиуса вершины. Во вложенном цикле мы создаем блоки и для каждого блока проходим по коллекции вершин, с целью определить вершины, принадлежащие каждому конкретному блоку. Каждую вершину мы оборачиваем в специально созданную для этого структуру Node:

public struct Node

{

public Coordinate coord;

public int number;

public int indexOfVertex;

public Node(Coordinate c, int number, int indexOfVertex)

{

this.coord = c;

this.number = number;

this.indexOfVertex = indexOfVertex;

}

}

где содержатся координаты вершины, ее порядковый номер и номер блока, которому она принадлежит. Последнее сделано для того, чтобы при поиске окружения вершины мы могли с легкостью выбирать все вершины, принадлежащие конкретному блоку. С этой же целью мы производим сортировку всех вершин по номеру блока, которому они принадлежат, чтобы максимально ускорить поиск вершин, принадлежащих одному и соседним блокам. Номер блока имеет следующий вид: j + i \* m, где i – номер строки, а j – номер столбца блока в соответствующей сетке блоков.

Array.Sort(ndmass, new SortByNumberOfBlockMin());

Целью запрограммированного алгоритма является сокращение времени поиска вершин графа, которые попадают в пределы радиуса окружности заданной вершины, по сравнении с прямым проходом по множеству всех вершин в графе. Используемый метод принимает два параметра среди которых координаты вершины и ее номер, возвращаемым же значением является массив номеров вершин, которые входят в радиус вершины – параметра функции:

public int[] Neighborhood(Coordinate x, int vertex);

В методе во вложенном цикле определяется в какой блок попадает вершина, после чего производится выборка вершин, которые принадлежат данному (блоку, в котором находится вершина-параметр) и соседним блокам. Для этой цели используем средства LINQ:

List<Node> highervertices = (from Node nd in ndmass

where (nd.number >= ((uint)(tmp - 1 - m)) && (nd.number <= ((uint)(tmp + 1 - m)))

)

select nd).ToList<Node>();

List<Node> vertices = (from Node nd in ndmass

where (nd.number <= ((uint)(tmp + 1)) && (nd.number >= ((uint)(tmp - 1))))

select nd).ToList<Node>();

// ищем вершины в блоках под основым

List<Node> lowvertices = (from Node nd in ndmass

where (nd.number >= ((uint)(tmp - 1 + m)) && (nd.number <= ((uint)(tmp + 1 + m))))

select nd).ToList<Node>();

Затем, среди этих вершин ищем те, которые попадают в радиус данной вершины-параметра.

1. Реализован специализированный класс для сбора статистики, содержащий метод для построения графа в формате CRS. На вход данного метода принимаются два целочисленных значения, которые указывают нижнюю и верхнюю границу числа ребер инцидентных каждой вершине, а также размер плоскости отображения и число вершин.

public void CreateTestCraph(int verticesamount, short powermin, short powermax, Size size);

Указанным методом создается тестовый граф, с заданным числом вершин, для каждой их которых число инцидентных ребер варьируется в пределах от нижней до верхней границы (принимаемые параметры), координаты которых генерируются случайным образом, что верно и для радиусов вершин графа.

// делаем массив, где будут храниться количества ненулевых элементов в каждой строке

int[] cntInRows = new int[verticesamount];

for (int p = 0; p < verticesamount; p++)

{

// устанавливаем число ненулевых элементов в строке, т.е

// фактически число ребер, которые есть у каждой вершины

cntInRows[p] = (int)rand.Next(powermin, powermax);

}

Заполняем CRS представление матрицы:

for (int i = 0; i < verticesamount; i++)

{

if (i != 0)

counter += cntInRows[i - 1];

// формируем номера столбцов в строке i

for (int j = 0; j < cntInRows[i]; j++)

// cntInRows[i] т.к для каждой строки свое число ненулевых элементов

{

do

{

graph.ColumnIndex[counter + j] = rand.Next(verticesamount);

f = false;

// заполняем разными числами, поэтому проверка на равенство

for (int k = 0; k < j; k++)

{

if ((graph.ColumnIndex[counter + j]) == (graph.ColumnIndex[counter + k]))

f = true;

}

}

while (f);

}

// сортируем номера столбцов в строке i

for (int t = 0; t < cntInRows[i] - 1; t++)

{

for (int k = 0; k < cntInRows[i] - 1; k++)

{

if ((graph.ColumnIndex[counter + k]) > (graph.ColumnIndex[counter + k + 1]))

int tmp = graph.ColumnIndex[counter + k];

graph.ColumnIndex[counter + k] = graph.ColumnIndex[counter + k + 1];

graph.ColumnIndex[counter + k + 1] = tmp;

}

}

}

}

Заполняем массивы вершин:

X = new double[verticesamount];

Y = new double[verticesamount];

for (int k = 0; k < verticesamount; k++ )

{

X[k] = Convert.ToDouble(randx.Next(size.Width \* 100)) / 100;

Y[k] = Convert.ToDouble(randy.Next(size.Height \* 100)) / 100;

}