САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ Математико-механический факультет

Курсовая работа

Реализация встроенного парсера языка DOT в библиотеку QuickGraph

Выполнила:

Студентка 242 группы

Говоруха Оксана

Научный руководитель:

старший преподаватель

Григорьев Семён Вячеславович

Содержание

1. Введение
2. Обзор
3. Постановка задачи
4. Реализация
5. Заключение
   1. Результаты
6. Список литературы

Введение

Для структурирования информации удобно использовать графы или диаграммы. Практически любые данные, в которых важны объекты и различные типы связей между ними хорошо моделируются графом. 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.

При работе с большими объемами данных уместно использовать специальные программы для их построения. С помощью библиотек для работы с графами можно реализовать огромное количество алгоритмов, которые могут иметь практическое применение: поиск кратчайшего пути, поиск циклов, проверка графа на связность и т. п.. Отдельные утилиты позволяют визуализировать графы, благодаря чему можно легче воспринимать ту структуру, с которой приходится работать и подробнее разобраться в связях.

Актуальной задачей является разработка более или менее универсальных библиотек, которые, с одной стороны, предоставляли бы пользователю высокоуровневые средства для работы с графами, а с другой, избавляли его от необходимости преобразований между различными внутренними представлениями графов.

Обзор

В настоящее время существует множество различных библиотек, предоставляющих средства для работы с графами. Они различаются не только внутренним представлением графов, но и набором встроенных алгоритмов для работы с ними, и средствами визуализации: некоторые библиотеки поддерживают только отрисовку графов, другие наоборот имеют слабые средства визуализации, но позволяют реализовывать большое количество алгоритмов для графов.

Среди существующих библиотек для работы с графами можно отметить следующие GrapX, JUNG, JGraphT и JGraph, QuickGraph, Boost, NetworkX, APSGraph, Arbor. Они написаны на разных языках и под разные платформы, каждая имеет как достоинства, так и недостатки.

GraphX for .NET - open-source библиотека для построения и визуализации графов, которая поддерживает множество различных алгоритмов раскладки графов и алгоритмов нахождения оптимальных путей ребер графа.

Boost представляет собой собрание библиотек классов, использующих [C++](https://ru.wikipedia.org/wiki/C%2B%2B) и предоставляющих удобный, кроссплатформенный интерфейс для работы с графами и не только.

APSGraph - объектно-ориентированная C++ библиотека для работы с ориентированными и неориентированными графами. Основной особенностью данной библиотеки является ориентация на максимальную производительность алгоритмов. В отличии от библиотеки Boost APSGraph не разделяет данные и алгоритмы по разным классам, что дает повышение производительности и инкапсуляцию данных. APSGraph содержит несколько алгоритмов, в частности такой не часто встречающийся алгоритм, как поиск всех циклов в ориетированном графе.

Библиотека [NetworkX](http://networkx.lanl.gov/) создана на языке Python и предназначенная для создания, манипуляции и изучения структуры, динамики и функционирования сложных сетевых структур. NetworkX имеет классы для работы с простыми, ориентированными и взвешенными графами, позволяет сделать узлом практически что угодно: текст, изображение, XML; поддерживает сохранение и загрузку графов в/из наиболее распространённых форматов файлов хранения графов; имеет встроенные процедуры для создания графов базовых типов, методы для обнаружения подграфов, К-дольных графов; может получать такие характеристики графа как степень вершин, высота графа, диаметр, радиус, длинна путей, центр. Также библиотека позволяет визуализировать сети в виде 2D и 3D графиков.

JGraphT - свободно распространяемая библиотека на языке Java, предоставляющая математические объекты и алгоритмы для работы с графами. JGraphT поддерживает различные типы графов и позволяет хранить в узлах любой тип. JGraphT и JGraph две разные библиотеки, которые предназначены для различных целей. JGraphT ориентирована на структуры данных и алгоритмов, а JGraph ориентирована на отрисовку и предоставления графического интерфейса для редактирования. Две библиотеки дополняют друг друга и могут быть использованы вместе с помощью JGraphModelAdapter предоставленной JGraphT.

JUNG- Java-библиотека, позволяющая моделировать, анализировать и визуализировать данные, которые могут быть представлены в виде графа или сети. Включает большое количество алгоритмов и платформу для легкой отрисовки сетевых данных.

QuickGraph является портативной библиотекой, которая предоставляет общие структуры для описания ориентированных, неориентированных графов и алгоритмы под .net, такие как измерение глубины дерева, поиск кратчайшего пути и т.п.. Библиотека имеет слабые средства визуализации, однако позволяет использовать сторонние программы MsAgl(GLEE) и Graphviz.

Рассмотрим отдельно один из распространенных пакетов утилит, предназначенных для автоматичесской визуализации графов - Graphviz. Для описания используется язык - DOT, структура такого графа представляет собой список субграфов, где каждый элемент - конструкция вида %имя% { }, в фигурных скобках содержатся комментарии и инструкции, описывающие граф. Такое представление хранится в текстовом файле с расширением .gv или .dot . Программа dot из пакета утилит Graphviz, принимая текстовое описание графа на выходе формирует граф в виде графического, векторного или текстового файла (.svg, .pdf, .jpg и т.п.). К сожалению, кроме визуализации в Graphviz нет возможности работать с самим графом или любой древовидной структурой. Конечно, можно использовать библиотеку QuickGraph для работы, а затем с помощью GraphViz его отразить, но если уже имеется описание графа на языке DOT, то переносить его вручную будет довольно громоздко, и при больших данных займет много времени, также не исключено, что можно допустить ошибку в графе. Если бы библиотека QuickGraph могла самостоятельно переносить граф с языка DOT, то это экономило бы время и решало проблему с возможными ошибками.

Поддержка библиотекой QuickGraph не только возможностей визуализации с помощью Graphviz, но и работы с dot-графами: преобразование графа, описанного на языке DOT, в новое представление с помощью структур QG, дает возможность применять алгоритмы библиотеки к другим структурам(описаниям графов) и, следовательно, расширяет ее функционал. Так как QuickGraph не поддерживает автоматическую загрузку и преобразование графа с языка DOT, то наличие встроенного парсера упрощало бы работу с ними.

Постановка задачи

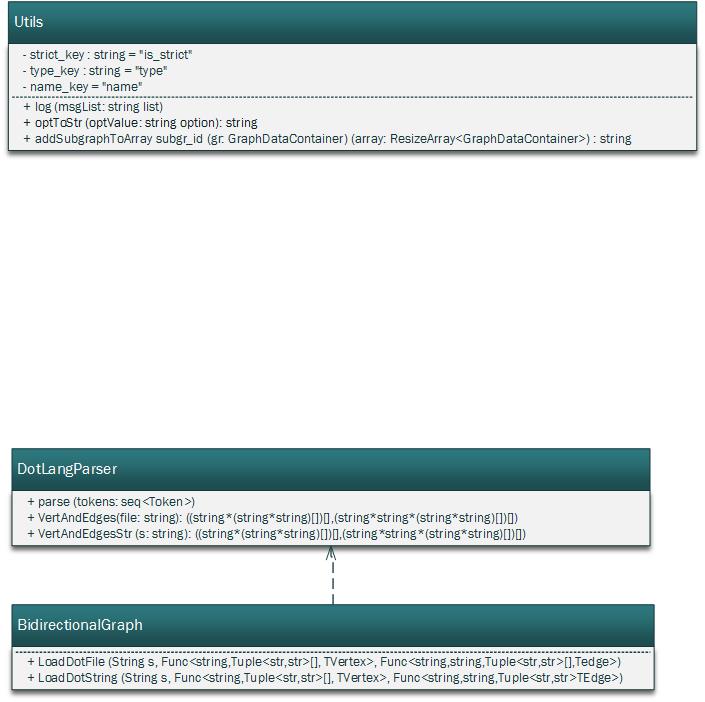
Основной целью работы является интегрирование парсера языка DOT в библиотеку QuickGraph. Для ее достижения должны быть поставлены следующие задачи:

1. Изучить документацию по библиотекам QuickGraph и GraphViz, сравнить различия реализации представлений и работы с графами
2. Реализовать метод в QuickGraph, позволяющий загружать файл с описанием графа на языке DOT или передавать описание, как аргумент
3. Встроить парсер в библиотеку QuickGraph и обеспечить получение на выходе нового представления графа с использованием структур библиотеки

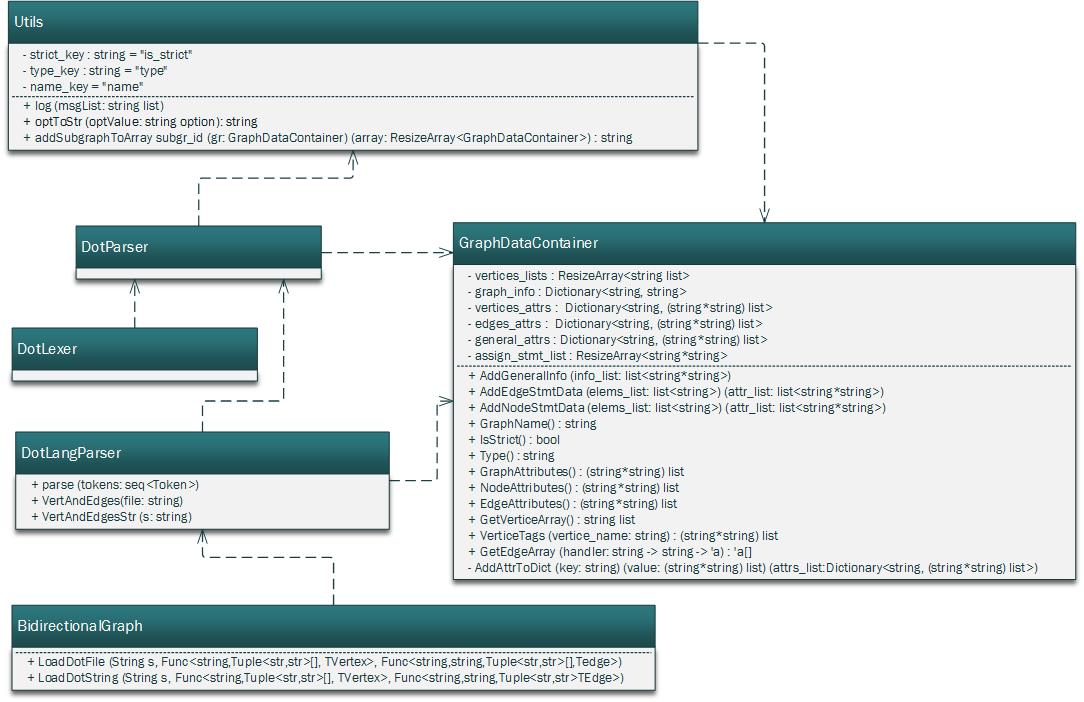
Реализация

Парсер языка DOT уже был написан, для дальнейшей с ним работы и возможности его встроить, надо чтобы он возвращал значения пригодные для интегрирования в библиотеку. В результате работы парсер собирает данные о поданном ему графе в тип GraphDataContainer, отдельно хранятся вершины, атрибуты ребер, атрибуты вершин, информация о самом графе, откуда в дальнейшем мы сможем их извлечь.

В QuickGraph реализовано большое количесво структур для графов, наиболее общими являются Bidirectional и Adjecency, но так как BidirectionalGraph хранит значение как о вершинах из которых исходит ребро, так и о тех, в которые входит, то было принято решение добавить метод в класс BidirectionalGraph<TVertex,TEdge>. Были реализованы два статических метода, которые автоматически инициализируют переменную и присваивают ей значение типа BidirectionalGraph<TVertex,TEdge>, один метод - для ввода пути к файлу с описанием графа на языке DOT, другой - для передачи самого описания графа в строке. Помимо файла методы должны принять функции, в которых будет объявлено тип нашей вершины и тип ребра. Внутри вызывается DotParser, которому передается только файл с описание нашего dot-графа, получаем две различные переменные, одна хранит массив вершин с их атрибутами, возвращаемый функцией VertWithAttrs или VertWithAttrsForStr в модуле DotLangParser, другая переменная хранит массив пар вершин с аттрибутами ребра, возвращаемый функцией EdgeWithAttrs или EdgeWithAttrsForStr. Для наглядности привожу часть полной диаграммы классов.



Ниже представлена полная диаграмма классов.



Вершины и ребра добавляются в BidirectionalGraph<TVertex,TEdge>, на выходе получаем исходный dot-граф представленный структурой из QuickGraph.

Например, зададим такие функции:

Func<string, Tuple<string, string>[], string> **func1** = (v, attrs) => v;

Func<string, string, Tuple<string, string>[], STaggedEdge<string, string>> **func2** = (v1, v2, attrs) => new STaggedEdge<string, string>(v1, v2, MyTaggs(attrs));

пусть исходный граф имеет такой вид:

strict graph test

{ 6 [lable = \"v1\"]

1[lable = \"v\"]

1 -- 2 [weight = 10]

2 -- 1

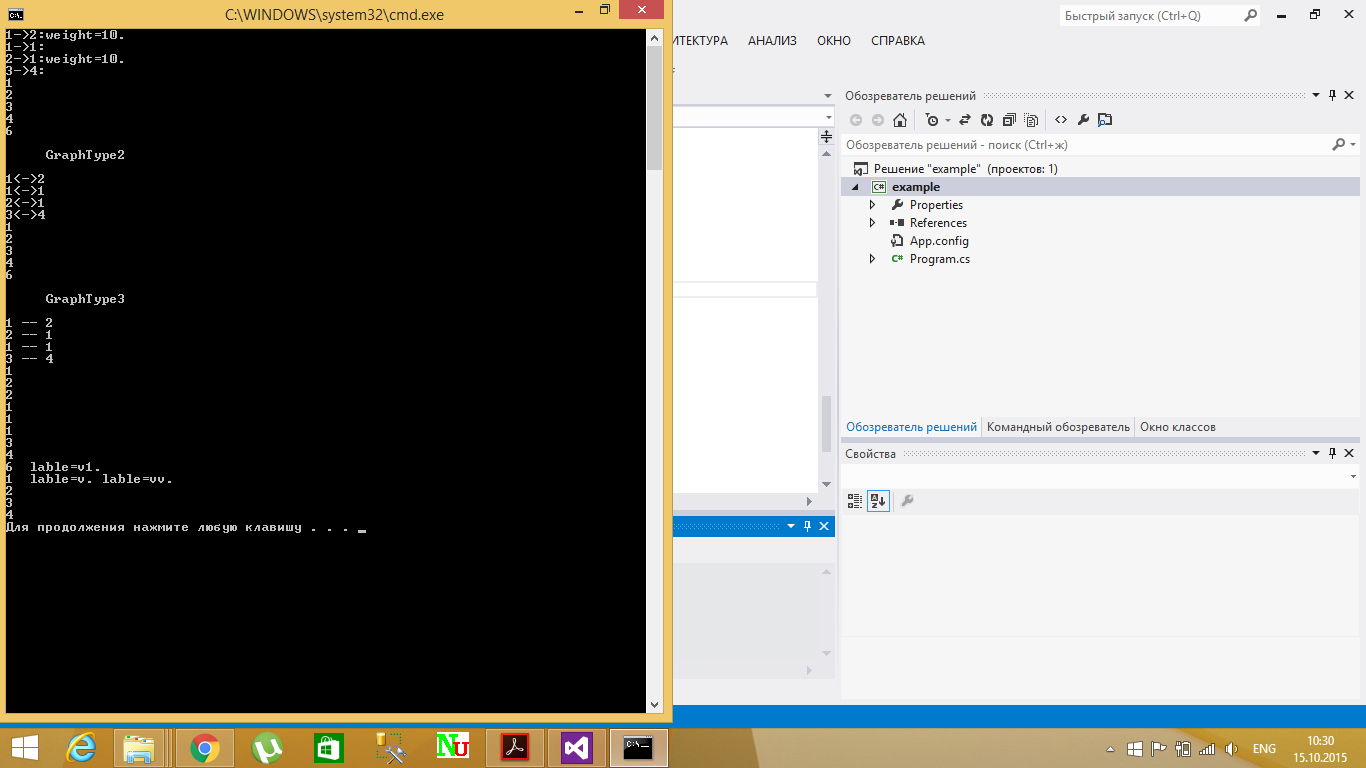
1 [lable = \"vv\"]

1 -- 1 [weight = 7]

3 --4

};

Вызовем метод LoadDotString:

var **g2** = BidirectionalGraph<string, STaggedEdge<string, string>>.LoadDotString(str, **func1**, **func2**);

Выведем на экран ребра и вершины графа, получим следующее представление.

Заключение

Таким образом, был реализован метод в классе BidirectionakGraph<TVertex,TEdge> библиотеки QickGraph, который вызывает парсер языка DOT и возвращает экземпляр этого класса.

Также написаны тесты и примеры, помогающие проверить правильность работы данной реализации.

Список литературы

1. Документация к библиотеке QuickGraph: <https://quickgraph.codeplex.com/documentation> ;
2. Документация к GraphViz: <http://graphviz.org/Documentation.php>;