Лабораторная работа №3 по курсу «Программирование (объектно-ориентированное программирование)»

Техническое задание

1. Постановка задачи

Написать программу на языке C++, реализующий алгоритм раскраски произвольного неориентированного графа. В ходе работы реализовать тип данных «Неориентированный граф» (UndirGraph).

2. Функциональный требования

2.1. Программа должна предоставлять пользователю возможность работать с неориентированным графом. Должна быть возможность создания графа как путем последовательных добавлений вершин графа и его ребер, так и путем автоматической генерации графа с заданным количеством вершин.

2.2. Программа должна по запросу пользователя осуществлять покраску графа. То есть каждая вершина должна быть условно помечена определенным «цветом» таким образом, чтобы цвета любых двух соседних вершин были разными.

2.3. Все методы основных типов данных должны быть покрыта модульными тестами. Должна по вводу пользователя проверять данные тесты.

2.4. Программа должна иметь консольный пользовательский интерфейс, имитирующий командную оболочку (в качестве ввода пользователю предлагается ввести команду для взаимодействия с программой).

3. Требования к типам данных

3.1. Для реализации основного типа данных – неориентированного графа – нужно реализовать более общий тип «ориентированный граф» (**Graph**). Затем нужно реализовать дочерний класс «неориентированный граф» (**UndirGraph**), у которого в общем случае переопределены 2 метода: добавления и удаления ребра.

3.2. Общий класс Graph, ориентированный граф, в качестве основной структуры хранения данных графа (вершин и ребер) импользует разреженную матрицу (Sequence<SparseSeq<unsigned int> *>). В качестве SparseSeq<T> можно использовать класс, реализованный в рамках предыдущей лабораторной работы. Данная матрица подразумевает под собой разреженную матрицу смежности графа. Также, объект типа Graph хранит поле со списком названий всех вершин (Sequence<std::string>).

3.3. Краткая спецификация класса Graph:

Название	Сигнатура	Описание
Атрибуты		
Разреженная матрица смежности	Sequence<SparseSeq<**unsigned int**> *> *matrix;	На пересечении i-ой строки и j-ого столбца матрицы хранится вес дуги между i-ой и j-ой вершины. Если дуги нет, хранится 0 (в смысле разреженной матрицы).
Список идентификаторов (имен) вершин	Sequence<string> *nodeNames;	Сопоставляет каждой i-ой вершине ее имя, выраженное в строковом типе.
Методы		
Конструктор	`Graph();`	Создает пустой граф (путая матрица смежности и пустой список строк).
Count	**int** nodeCount();	Возвращает количество вершин в графе (пустой граф – 0 вершин)
AddNode	**void** addNode(string node);	Добавляет вершину с заданным именем в граф. В случае существования в графе вершины с таким выбрасывается исключение
RemoveNode	**void** removeNode(string node);	Удаляет вершину с заданным именем. Выбрасывает исключение в случае отсутствия такой вершины.
AddEdge	**virtual void** addEdge(string node1, string node2, **unsigned int** weight);	Добавляет ребро между двумя вершинами. Выбрасывается исключение, если хотя бы одного из ребер не существует.
RemoveEdge	**virtual void** removeEdge(string node1, string node2);	Удаляет ребро между двумя вершинами. Выбрасывается исключение, если хотя бы одного из ребер не существует.
HasEdge	**bool** hasEdge(string node1, string node2);	Возвращает true, если существует ребро между двумя вершинами, иначе false
HasNode	**bool** hasNode(string node);	Возвращает true, если в графе есть вершина с заданным именем, иначе false
Nodes	Sequence<string> *listOfNodes();	Возвращает список имен вершин (геттер для nodeNames)
GetNeighbours	Sequence<string> *getNeighbours(string nodeName);	По заданной вершине возвращает список соседей для нее.
NodeName	string nodeName(**int** index);	По индексу возвращает имя вершины.

3.4. Класс «ориентированный граф» (UndirGraph) получается путем наследования от класса Graph. При этом в классе UndirGraph переопределяются 2 метода: AddEdge и RemoveEdge, ответственные за добавление и удаление ребер. Подразумевается, что при добавлении/удалении ребра оно добавляется/удаляется, как ориентированное ребро в обе стороны. Таким образом краткая спецификация данного класса будет иметь такой же вид, как и для класса Graph (т.к. данный класс является дочерним), но с переопределенными методоми AddEdge и RemoveEdge.

3.5. Последней структорой данных является класс ColorizedGraph, инкапсулирующий в себе непосредственно алгоритм покраски графа. Класс «покрашенный граф» является дочерним для UndirGraph, так как покрашенным граф в нашем случае может быть только если он является неориентированным. Помимо полей характерных неориентированному графу, такой класс имеет поле Sequence<Sequence<string> *> *coloredNodes, представляющее собой распределение вершин по цветам. Индекс в данном списке представляет собой отдельный цвет, значение – список вершин, покрашенных в данных цвет. Конструктор в данном классе имеет следующую сигнатуру: **explicit** ColorizedGraph(UndirGraph *graph). На основе объекта типа UndirGraph инициализируются поля, в частности поле coloredNodes. Таким образом внутри конструктора происходит действие алгоритма по покраске вершин графа.

4. Реализация алгоритма

Алгоритм основывается на действиях над матрицей смежности графа. Для этого строится уже не разреженная матрица смежности графа, так как в дальнейшем смысл разреженности пропадает (большинство строк графа будут заполнены единицами).

Матрица смежности будет представлять из себя матрицу N * N, где N – количество вершин, заполненную нулями и единицами (в нашей реализации – булевы значения). Условимся для данного алгоритма, что на диагонали стоят единицы.

Начиная с первой строки матрицы смежности, будем идти по всем «невычеркнутым» строкам (множество невычеркнутых строк вначале пустое). Для каждой i-ой невычеркнутой строки идем по ее элементам вплоть до первого нулевого элемента. Как только такой элемент встретился (пусть индекс j), проверяем, является ли j-ая строка исключенной. Если является – пропускаем, идем дальше, если нет - прибавляем побитово (то есть с помощью поэлементной дизъюнкции) к текущей строке строку с индексом j. После этого добавляем j-ую строку в множество исключенных вершин. Формируем цветовую группу из i-ой и j-ой строки. После этого повторяем данную

процедуру, каждый раз добавляя к цветовой группе новые строки, до тех пор, пока в i-ой строке либо не будут все единицы, либо пока все остальные строки не окажутся исключенными.

Таким образом будут пройдены все (не исключенные в процессе алгоритма) строки матрицы смежности и сформированы все цветовые группы.