**Лабораторная работа №3**

**по курсу «Языки программирования и методы программирования» (информатика, 3 семестр)**

**Техническое задание**

1. **Постановка задачи**

Написать программу на языке С++, реализующий алгоритм раскраски произвольного неориентированного графа. В ходе работы реализовать тип данных «Неориентированный граф». Использовать ранее написанный АТД «Словарь», так как он наиболее удобен для списков смежности (вершина: смежные с ней). Вершины графа могут быть любым типом данных.

1. **Функциональные требования**
   1. АТД “Неориентированный граф” должен позволять хранить значения любых типов. Для этого класс UndirectedGraph должен быть объявлен как шаблонный. Пример:

template<typename L, typename N>

class UndirectedGraph{

private:

unsigned int edgesQuantity{};

unsigned int nodesQuantity{}

...

private:

dictionary::IDictionary<N, Node<N>\*> \*nodes;

map<pair<N,N>,L> edges

public:

SparseVector(...);

…

}

L – это параметр, обозначающий тип веса.

N - это параметр, обозначающий тип вершины.

* 1. ТД «Неориентированный граф» должен обладать, по крайней мере, следующими атрибутами и методами:
  2. Атрибуты:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Название** | **Сигнатура** | **Назначение** |
|  | edgesQuantity | unsigned int edgesQuantity | количество ребер |
|  | nodesQuantity | unsigned int nodesQuantity | количество вершин |
|  | nodes | dictionary::IDictionary<N, Node<N> \*> \*nodes | словарь, содержащий вершину: смежные с ней |
|  | edges | std::map<std::pair<N, N>, L> edges; | хранение ребер и их веса |

* 1. Методы (операции):

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Название** | **Сигнатура** | **Назначение** |
|  | addNode | void addNode(N node); | Добавление вершины |
|  | addEdge | void addEdge(N node1, N node2, L label); | Добавление ребра |
|  | degreeOfNode | unsigned int degreeOfNode(N node); | Получение степени вершины |
|  | checkPath | bool checkPath(N node1, N node2); | Проверка, соединены ли вершины |
|  | toString | std::string toString(); | Получение строки из графа, для визуализации |
|  | isNeighbours | bool isNeighbours(N node1, N node2); | Проверить, являются ли вершины “соседями” |
|  | getLabel | L getLabel(N node1, N node2); | Получить вес ребра |
|  | removeNode | void removeNode(N node); | Удалить вершину |
|  | giveColors | dictionary::AVLTreeDictionary<int, myarray::ArraySequence<N>> giveColors(); | Раскрасить граф |
|  | removeEdge | void removeEdge(N node1, N node2); | Удалить ребро |

:

1. **Требования к структурам данных и алгоритмам**
   1. Должны быть организованы следующие структуры данных:

* «Неориентированный граф» UndirectedGraph
* Получение размера разреженного вектора
* Добавление вершины
* Добавление ребра
* Удаление вершины
* Удаление ребра
* Раскраска дерева
* Проверка, существует ли путь между вершинами
* Вывод разреженного вектора

1. **Требования к интерфейсу**

Реализовать консольное приложение для демонстрации работы UndirectedGraph и раскраски графа

1. **Требования к форматам входных и выходных данных**
   1. **Требования к форматам выходных данных**

Выводить граф в формате Node1 <--> Node2 label: 1, Node2 <--> Node3 label: L2….

Выводить раскрашенный графа в формате [color: Node1 Node2, …, colorN: NodeN].

1. **Требования к Unit-тестам**

Покрыть Unit-тестами