Лабораторная работа №2

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

Техническое задание

Используемые термины и сокращения

|  |  |
| --- | --- |
| АТД | Абстрактный тип данных |
|  |  |
|  |  |

# Постановка задачи

На языке C++ реализовать АТД «Ориентированный граф», «Неориентированный граф» на основе АТД «Матрица», реализованной на основе АТД «Последовательность». Решить задачу поиска кратчайшего пути на графе и топологического упорядочивания. Написать unit-тесты. Реализовать пользовательский интерфейс.

# Функциональные требования

* 1. АТД «Ориентированный граф» и «Неориентированный граф» должны позволять хранить в качестве весов ребер графа любые численные типы данных. Для этого классы должны быть объявлены как шаблонные.
  2. АТД «Неориентированный граф» должен быть реализован как наследник класса «Ориентированный граф».
  3. АТД «Ориентированный граф» («DirGraph») должен обладать, по крайней мере, следующими методами:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Название** | **Сигнатура** | **Назначение** |
|  | changeEdge | void changeEdge(int vert1, int vert2, T weight) | Изменить или добавить новое ребро между существующими вершинами |
|  | getSize | int getSize() | Получить количество вершин в графе |
|  | toString | string toString() | Преобразовать граф в строку. Как правило, для вывода. |
|  | getPath | ArraySequence<int>\* getPath(int vert1, int vert2) | Получить кратчайший путь от вершины к вершине. |
|  | getPathWeight | T getPathWeight(ArraySequence<int> \*path) | Получить вес некоторого пути. |
|  | getTopologicalSort | ArraySequence<int>\* getTopologicalSort() | Получить топологическое упорядочивание графа. |

* 1. АТД «Неориентированный граф» («Graph») должен наследовать все методы класса «Ориентированный граф» («DirGraph») кроме метода getTopologicalSort. При попытке вызова этого метода должно выбрасываться исключение. Метод changeEdge должен иметь отличную реализацию. Отличие должно заключаться в заполнении матрицы смежностей. В неориентированном графе матрица должна оставаться симметричной.
  2. Метод changeEdge должен выбрасывать исключение, если идет попытка создать ребро между некоторой вершиной и ей самой.
  3. Метод getPathWeight должен выбрасывать исключение, если в качестве параметра передан невозможный путь.

# Требования к структурам данных и алгоритмам

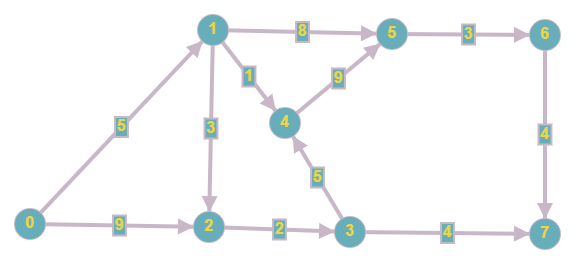
* 1. АТД Ориентированный граф» и «Неориентированный граф» должны быть основаны на матрице смежности графа.
  2. Метод «getPath», решающий задачу поиска кратчайшего пути в графе между двумя вершинами с учетом весов ребер, должен быть основан на алгоритме Дейкстры. Этот метод не должен, в отличие от алгоритма Дейкстры, искать кратчайшие пути до всех вершин графа.
  3. Метод getTopologicalSort, решающий задачу топологического упорядочивания графа должен быть основан на алгоритме обхода графа в глубину.

# Требования к интерфейсу

## Должен быть реализован консольный пользовательский интерфейс с возможностью тестирования реализованных алгоритмов. Должна иметься возможность тестировать алгоритмы как на ориентированных, так и на неориентированных графах.

## В интерфейсе должна быть возможность использования стандартного тестового графа, а также создания пользовательского графа произвольного размера.

### Стандартный тестовый граф:



### Стандартный тестовый неориентированный граф должен быть аналогичен показанному на рисунке с тем отличием, что все ребра являются ненаправленными.

## В интерфейсе должна быть возможность поиска кратчайшего пути между двумя точками на графе. Интерфейс должен показывать и путь, и вес этого пути.

# Требования к форматам входных и выходных данных

Требования к форматам входных данных

Нет.

Требования к форматам выходных данных

Нет.

# Требования к unit-тестам

* 1. Написать, по крайней мере, один тест, реализующий следующий сценарий:
     1. Задан стандартный тестовый граф
     2. Строится путь из 6 вершины в 5.
     3. Проверяется, что размер полученной последовательности равен 1
     4. Проверяется, что нулевый элемент равен -1, то есть такого пути не существует.
     5. Строится путь из вершины 0 в вершину 2
     6. Проверяется, что размер полученной последовательности равен 3
     7. Проверяется, что нулевой элемент это 0
     8. Проверяется, что первый элемент это 1
     9. Проверяется, что второй элемент это 2
     10. Проверяется, что вес найденного пути равен 8
  2. Написать, по крайней мере, один тест, реализующий следующий сценарий:
     1. Задан стандартный тестовый неориентированный граф
     2. Строится путь из 6 вершины в 5.
     3. Проверяется, что размер полученной последовательности равен 2
     4. Проверяется, что нулевой элемент равен 6
     5. Проверяется, что первый элемент равен 5
     6. Проверяется, что вес найденного пути равен 3
     7. Строится путь из вершины 0 в вершину 2
     8. Проверяется, что размер полученной последовательности равен 3
     9. Проверяется, что нулевой элемент это 0
     10. Проверяется, что первый элемент это 1
     11. Проверяется, что второй элемент это 2
     12. Проверяется, что вес найденного пути равен 8