Лабораторная работа №3

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 структуру данных ”граф” и алгоритмы на графе: выявление циклов на графе, топологическая сортировка вершин графа, алгоритм Дейкстры, построение транзитивного замыкания графа .

Решения: граф хранится в виде списка вершин, каждая вершина хранит список исходящих из нее дуг. Дуга хранит номер узла, из которого она исходит, вес дуги и номер узла, в который она входит. Также граф хранит информацию о том, является он направленным или нет.

Алгоритмы:

Обход в глубину:

1. Выбирается начальная вершина и посещаются;
2. При посещении вершины: Новую вершину добавляем в множество посещенных. Дуги новой вершины добавляем в стек. Не добавляем дуги, которые ведут в уже посещенные вершины. При выборе следующей для посещения, выбираем первую дугу в стеке. Критерий останова: стек пуст и нет не посещённых вершин.

Поиск цикла на графе:

Граф обходится в глубина, за тем лишь исключением, что дуги ведущие в уже посещённые вершины добавляются в очередь вершин, и если посещается дуга, находящаяся в множестве посещенных вершин, то алгоритм останавливается и выводит знак наличия цикла.

Топологическая сортировка вершин графа:

Создается массив посещенных вершин и стек, в котором будут хранится отсортированные вершины.

Граф обходится в глубину, начальной точкой поочередно выбираются все вершины, за тем лишь исключением, что при посещении новой вершины сначала вызывается функция посещения вершин, смежных с посещенной, а потом уже вершина добавляется в стек.

Алгоритм Дейкстры:

0)Создаются два массива(для хранения посещенных вершин и для хранения весов вершин)

1)Всем вершинам присваивается вес (вес начальной вершины 0, остальным недостижимо большой, относительно весов дуг).

2)Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещённых вершин выбирается вершина u, имеющая минимальную метку. Мы рассматриваем всевозможные маршруты, в которых u является предпоследним пунктом. Вершины, в которые ведут рёбра из u, назовём соседями этой вершины. Для каждого соседа вершины u, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки u и длины ребра, соединяющего u с этим соседом. Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину u как посещённую и повторим шаг алгоритма.

Транзитивное замыкание графа:

Для каждой вершины графа выполняется следующий алгоритм:

Происходит обход графа в глубину, для каждой новой посещенной вершины строятся дуги, входящие в эту вершину и исходящие из вершин посещенных до этого.

# Функциональные требования

* 1. класс должен быть объявлен как шаблонный. Пример:

template<class T>

class Graph

{

private:

int Vcnt, Ecnt;

bool digraph;

struct **node** {

ArraySequence<Edge<T>\*>\* adj;

int num;

**node**(int number)

{

num = number;

adj = new ArraySequence<Edge<T>\*>;

}

};

ArraySequence<node\*>\* Nodes;

…

}

* 1. Класс дуги:

template<class T>

class **Edge** {

public:

**Edge**(int f, int t, T w)

{

from = f;

to = t;

wt = w;

};

**Edge**()

{

from = -1;

to = -1;

wt = -1;

};

T wt;

int from;

int to;

};

АТД «Граф» («SparsedVector») должен обладать, по крайней мере, следующими атрибутами и методами:

* 1. Атрибуты

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Название** | **Сигнатура** | **Назначение** |
|  | Vcnt | int V() | количество вершин |
|  | Ecnt | Int E() | Количество дуг |

* 1. Методы (операции):

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Название** | **Сигнатура** | **Назначение** |
|  | Get | ArraySequence<Edge<T>\*>\* Get(int v) | Получение списка дуг, исходящих из v-ой вершины. |
|  | RemoveNode | void RemoveNode(int v) | Удаление узла |
|  | RemoveEdge | void **RemoveEdge**(int from, int to) | Удаление дуг соединяющих два элемента |
|  | **insertNode** | void **insertNode**(node\* n) | Добавление вершины |
|  | **insertEdge** | void **insertEdge**(Edge<T>\* e) | Добавление дуги. |
|  | **tc** | void **tc**() | Функция дополнения графа до транзитивного |
|  | **tcR** | void **tcR**(int v, int w, T dist) | Рекурсивная функция дополнения подграфа до транзитивного |
|  | **edge** | bool **edge**(int v, int w) | Функция нахождения дуги на графе |
|  | **AllCyclic** | bool **AllCyclic**() | Функция нахождения цикла на графе |
|  | **cyclic** | bool **cyclic**(int v, int& cycle\_st, int color[]) | Функция нахождения цикла в подграфе |
|  |  |  |  |
|  | **topologicalSort** | Stack<T>\* **topologicalSort**() | Топологическая сортировка |
|  | **Dijkstra** | ArraySequence<T>\* **Dijkstra**(int v) | Алгоритм Дейкстры |

# Требования к структурам данных и алгоритмам

* 1. Для хранения данных используются:
     1. Stack<T>
     2. ArraySequence< T>
     3. Выполнить реализацию, использующую хэш-таблицы (IDictionary), и реализацию, использующую отсортированную последовательность (ISortedSequence). Сделать оба эти класса наследниками абстрактного класса DataSparsedVector.
     4. Поиск по отсортированной последовательности и вставка в нее должны выполняться за логарифмическое время.
     5. Поиск по хэш-таблице и вставка в нее должны выполняться за константное время.

# Требования к интерфейсу

Графический интерфейс должен позволять пользователю: автоматически или вручную вводить данные, получать выходные данные в виде текстового вывода на экран.

# Требования к форматам входных и выходных данных

Требования к форматам входных данных

Нет.

Требования к форматам выходных данных

Нет.

# Требования к unit-тестам

проверить все методы класса Graph