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 «Разреженный вектор». Обеспечить хранение данных в виде (индекс в векторе- элемент) в отсортированной последовательности или хэш-таблице.

Решения: при создании разреженного вектора, в него добавляются только “ненулевые” элементы(“ненулевыми” элементами будем называть элементы удовлетворяющие какому-то условию. Для типа данных int это элементы не равные нулю). Элемент из входных данных характеризуется ключом(например индекс в массиве). Также можно запоминать диапазон индексов, чтобы удобнее осуществлять переход от разреженного вектора к обычному.

# Функциональные требования

* 1. АТД должен позволять хранить элементы любых типов. Для этого класс должен быть объявлен как шаблонный. Пример:

template<class Key, class Value>

class SparsedVector

{

private:

DataSparsedVector<Key, Value>\* Data;

int Length;

…

int GetLength();

< Value > Get(Key key);

void Add(Key key, Value value)

…

}

Value – это параметр, обозначающий тип элемента[[1]](#footnote-1).

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

* 1. Атрибуты

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Название** | **Сигнатура** | **Назначение** |
|  | Length | int getLength() | Длина последовательности (количество элементов) |
|  | Data | DataSparsedVector<Key, Value>\* Data | АТД для хранения данных |

* 1. Методы (операции):

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Название** | **Сигнатура** | **Назначение** |
|  | Get | Value Get(Key key) | Получение элемента по индексу. |
|  | Remove | void Remove(Key key) | Удаляет элемент из последовательности. Если элемента не было, ничего не происходит. |
|  | ContainsKey | Bool ContainsKey(Key key) | Проверка наличия элемента в векторе по ключу |
|  | Add | void Add(Key key, Value value) | Добавление элемента в вектор |
|  | istrue | bool istrue(int value) | Возвращает истину если элементы не нулевой и лож в противном случае. |

# Требования к структурам данных и алгоритмам

* 1. Для хранения данных реализовать следующие варианты:
     1. Использовать контейнер IDictionary
     2. Использовать контейнер ISortedSequence
     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: выбирать тип данных, который будет осуществлять хранение вектора(отсортированная последовательность или хэш-таблица), 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 SparsedVector

1. В первом приближении можно выполнить «обычную» реализацию, на примере какого-либо простого типа элементов, скажем, int. Обобщение до поддержки любого типа будет почти механическим. [↑](#footnote-ref-1)