Лабораторная работа №2

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

Техническое задание

|  |  |  |
| --- | --- | --- |
| Дата | 07.12.2016 |  |
| Выполнила | Елагина Д. |  |

Используемые термины и сокращения

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

# Постановка задачи

Написать программу на C++ для сравнения различных алгоритмов поиска. Выполнить реализацию. Написать для нее тесты.

# Функциональные требования

* 1. АТД должен позволять хранить элементы любых типов. Для этого класс должен быть объявлен как шаблонный.
  2. Данная последовательность должна сортироваться тремя способами:
* Сортировка с помощью простого выбора
* Сортировка с помощью выбора из дерева
* Быстрая сортировка
  1. Следует рассматривать три основных случая:
* массив уже отсортирован в нужном направлении
* массив отсортирован в обратном направлении
* массив не отсортирован
  1. Программа должна предоставлять функцию измерения времени выполнения алгоритма. Должна быть функция сравнения алгоритмов – по времени выполнения на одних и тех же входных данных.

# Требования к структурам данных и алгоритмам

* 1. Для хранения данных реализовываем:
     1. Выполнить реализацию, использующую массивы (ArraySequence), и реализацию, использующую связные списки (ListSequence). Сделать оба эти класса наследниками абстрактного класса Sequence для хранения элементов в последовательности.
     2. Выполнить реализацию, использующую бинарные деревья (BinaryTree), и реализацию, использующую связные B-деревья (BTree). Сделать оба эти класса наследниками абстрактного класса Tree для хранения элементов в деревьях.
  2. Все методы, принимающие на вход индексы, должны выбрасывать исключение в ситуациях, когда индекс выходит за допустимые границы: меньше 0 или больше или равен числу элементов в последовательности.
  3. Поддержка дополнительного интерфейса SortedSequence позволяет внутри класса Tree получить индекс элемента в дереве (IndexOf) или получить подпоследовательность (поддерево) существующего дерева (GetSubsequence). IndexOf возвращает индекс первого встретившегося в дереве элемента, ключ которого совпадает с заданным.
  4. Деревья поиска реализовываны как шаблонные классы. Это позволяет как хранить в них сами данные, так и лишь индексы, отмечающие положение данных в последовательности. Т.е. у шаблона два параметра-типа: тип ключа и тип хранимого значения.
  5. Алгоритм построения дерева из последовательности реализован в рамках отдельного класса – TreeBuilder.
  6. Программа предоставляет функцию измерения времени выполнения алгоритма на трех видах входных данных:
* Произвольная последовательность
* Последовательность, отсортированная в правильном порядке
* Последовательность, отсортированная в обратном порядке.

# Требования к интерфейсу

Программа должна обладать пользовательским интерфейсом (консольным или графическим). При этом должна быть возможность как автоматической, так и ручной проверки корректности работы алгоритмов.

# Требования к форматам входных и выходных данных

Требования к форматам входных данных

Программа должна позволять выбрать любой из реализованных алгоритмов сортировки и запустить его на (достаточно произвольных) исходных данных.

## Требования к форматам выходных данных

Должна быть возможность просмотра как исходных данных, так и результата.

# Требования к unit-тестам

* 1. Тестирование бинарного дерева
     1. Вывод пустого дерева
     2. Вставка 1
     3. Вставка 3
     4. Вставка 2
     5. Индекс 0 (ожидаем -1)
     6. Индекс 1 (ожидаем 0)
     7. Индекс 2 (ожидаем 1)
     8. Индекс 3 (ожидаем 2)
     9. Индекс 4 (ожидаем -1)
     10. Подпоследовательность (-1, -1) (ожидаем исключение – отрицательный индекс)
     11. Подпоследовательность (1, 4) (ожидаем исключение – индекс больше длины последовательности)
     12. Подпоследовательность (1, 2) (ожидаем дерево 2 3)
     13. Очистить память
  2. Тестирование B-дерева
     1. Вывод пустого дерева
     2. Вставка 1
     3. Вставка 3
     4. Вставка 2
     5. Индекс 0 (ожидаем -1)
     6. Индекс 1 (ожидаем 0)
     7. Индекс 2 (ожидаем 1)
     8. Индекс 3 (ожидаем 2)
     9. Индекс 4 (ожидаем -1)
     10. Подпоследовательность (-1, -1) (ожидаем исключение – отрицательный индекс)
     11. Подпоследовательность (1, 4) (ожидаем исключение – индекс больше длины последовательности)
     12. Подпоследовательность (1, 2) (ожидаем дерево 2 3)
     13. Очистить память
  3. Закончить работу.