Лабораторная работа № 2

по курсу “Языки программирования и методы программирования” (информатика, 3 семестр)

Техническое задание

1. Постановка задачи

Написать программу на C++ для сравнения различных алгоритмов поиска. Сравнение алгоритмов должно производиться на одной из приведенных задач, связанных с обработкой информации. Выполнить реализацию. Написать для нее тесты. Реализовать пользовательский интерфейс.

2. Функциональные требования

Задачи, выполняемые в данной работе: поиск наиболее частой подпоследовательности, размер которой задается пользовательским диапазоном целых чисел, в строке; обработка разряженных векторов; построение алфавитного указателя, где количество слов в строке задается пользователем. Все задания на ассоциативную память.

Задачи должны быть выполнены с помощью реализации шаблонного класса IDictonary, построенного на бинарном дереве, и классе Sequence, написанного в предыдущих лабораторных работах.

Класс SearchMaxSub должен находить наиболее часто повторяющуюся подпоследовательность размера, укладывающимся в пределы, указанные пользователем, и уметь выводить ее, ее длину, пару строка - длина в качестве ответа.

Принцип работы: циклический проход по последовательности для проверки каждой находящейся в заданном диапазоне длин подпоследовательностей, подсчет количества их встреч в значении для ключа, которым является сама подпоследовательность, и одновременное запоминание чаще всего встречающейся подстроки и количества встреч в отдельных переменных.

Основные методы:

SearchMaxSub()	Поиск максимально встречающейся подпоследовательности	Сложность: O(n^4)

GetMaxCount()	Возвращает число - количество раз, которое встретилась найденная подпоследовательность	O(1)
GetMaxString()	Возвращает саму подпоследовательность	O(1)

Класс Sparce должен обрабатывать разряженные векторы, то есть удалять нулевые элементы, сохраняя индексацию для ненулевых элементов, а также выводить как разряженные, так и обработанные вектора.

Принцип работы:

Создание нового вектора на основе старого за счет удаления нулевых элементов с сохранением индексов оставшихся элементов благодаря сохранению в хэш-таблице.

Основные методы:

Sparce()	Создает обычный вектор в виде словаря, где ключ - индекс элемента в изначальном векторе	Сложность: O(nlog(n))
GetSparceVector()	Возвращает полученные вектор	O(nlog(n))
GetSparceSize()	Возвращает размер полученного вектора	O(1)

Класс Split_into_pages разбивает текст на страницы, заданного размера, умеет выводить отдельные страницы, весь обработанный текст, количество страниц, полученных в “новом тексте”, страницу, на которой содержится конкретное слово.

Принцип работы:

Связывает слово, являющейся ключом в словаре, со страницей в качестве его значения. Разброс слов по страницам происходит во вложенном цикле, который учитывает не только пользовательское пожелание для разбиения текста, но и заданные в самой задаче требования.

Основные методы:

Split_into_pages()	Разбивает текст на страницы по заданным параметрам	Сложность: O((n^2)log(n))
GetPage()	Возвращает часть словаря, отвечающую за конкретную страницу	O(nlog(n))
PrintHowPage()	Печатает номер страницы, на которой находится искомое слово	O(log(n))

3. Требования к структурам данных

IDictonary:

Хеш-таблица должна хранить данные в структуре данных, обеспечивающих обращение по индексу. Хеш-функция должна вычислять положение в этом массиве на основании ключа, а также проверять наличие элемента в таблице по ключу. При удалении элементов в таблице должны отсутствовать пустоты. Таблица обязана поддерживать добавление новых элементов и знать о количестве элементов, хранимых в ней.

Методы:

GetCount()	Возвращает количество элементов в словаре	Сложность: O(1)
Get()	Возвращает элемент	O(log(n))
ContainsKey()	Проверяет на истину нахождение элемента в словаре	O(log(n))
Add()	Добавляет элемент в словарь	O(log(n))
Remove()	Удаляет элемент из словаря	O(log(n))

BinTree

Класс сбалансированного бинарного дерева поиска должен поддерживать удаление, добавление, поиск и проверку наличия элементов в структуре

данных, нахождение поддерева, поиск минимального и максимального элементов, вывод в консоль, удаление всего дерево и запоминание его размера.

Основные методы:

Insert()	Вставляет элемент в дерево согласно определению бинарного дерева поиска и после делает балансировку: поворот в правую или левую сторону в зависимости от разности показания весов у узлов	Сложность O(log(n))
Search()	Ищет узел с заданным ключом и возвращает найденный узел	Сложность O(log(n))
delete_()	Рекурсивно удаляет элемент из дерева. Если удаляется корень, то его значение меняется местами с наименьшим и далее идет его удаление из узла правого поддерева.	Сложность O(log(n))

pair_

Класс “пара” должен поддерживать операторы сравнения для различных пар, вывод в консоль, конструктор по ключу и значению, пустой конструктор и по ссылке на другую пару.

Сравнение происходит по ключу. Сложность всех функций - константная.

Sequence

Класс, являющийся “интерфейсом” классов ArraySequence и LinkedListSequence, должен поддерживать такие функции как получение значения по индексу, вставка в любое место последовательности,

изменение размера, получения подпоследовательности, удаление элемента, добавление или замена элемента по индексу.

Основные методы:

GetSize()	Возвращает количество элементов в последовательности	Сложность: O(1)
Get()	Возвращает элемент по индексу	O(n)
GetSubList()	Возвращает последовательность в заданном диапазоне индексов	O(n)
Delete_()	Удаляет элемент по индексу	O(n)
Set()	Вставляет элемент по заданному индексу	O(1)

4. Требования к интерфейсу

Интерфейс должен предоставлять пользователю выбор между задачами, поддерживать как автоматический, так и ручной ввод данных, давать возможность пользователю увидеть результат и промежуточные значения.

5. Требования к входным и выходным данным

Данные должны являться символами или целыми числами в зависимости от выбранной задачи.

6. Требования к тестированию

Все основные алгоритмы должны быть покрыты тестами.