Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу дискретного анализа

Студент: А. В. Синявский
Преподаватель: Н. А. Зацепин
Группа: М8О-308Б-18
Дата:
Оценка:
Подпись:

Москва, 2020

Лабораторная работа № 2 по курсу дискретного анализа

Выполнил студент группы М80-208Б-18 МАИ *Синявский Андрей.*

Условие

1. Необходимо создать программную библиотеку, реализующую указанную структуру данных, на основе которой разработать программу-словарь. Ключ - регистронезависимые слова не длиннее 256 символов, значение - числа в диапазоне от 0 до $2^{64} - 1$

2. Вариант 03.

   PATRICIA.

Метод решения

PATRICIA по своей структуре крайне похожа на обычное бинарное дерево поиска, только с нюансами вроде сравнения битов и наличия обратных указателей. Всю программу я для себя разделил на 2 условные части: сама патриция, и операции Save/Load, которые лично мне удобнее рассматривать отдельно от остальной программы.

1. PATRICIA.

   Необходимо реализовать вектор, который будет использован в качестве строк, в которых будут храниться слова-ключи. Нужно определить структуру узла, в котором должны быть поля для ключа, значения, битового поля, а также указатели на потомков (или обратные указатели). Для самой структуры данных нужно написать операции вставки, поиска и удаления элемента, которые я буду реализовывать по книге HANDBOOK OF DATA STRUCTURES AND APPLICATIONS за авторством Dinesh P. Mehta и Sartaj Sahni.

2. Save/Load.

   Сохранение будет представлять собой рекурсивный обход дерева как бинарного, с оглядкой на возможность обратных указателей, и запись данных каждого узла в указанный в аргументе файл. Также, для определения формата файлов сохранения, в файл будет записан ключ, наличие которого будет проверяться при загрузке.

   Функция загрузки же будет просматривать файл, созданный функцией сохранения, заполнять поля нового узла прочитанными данными, и вставлять этот узел в дерево, схожим образом с операцией вставки в дерево.

Описание программы

Узлы представлены объектами класса TNode, патриция объектом класса TPatricia. Поля и методы этих классов не нуждаются в дополнительном пояснении. За пределы классов вынесены функции BitGet и LeftMost. Первая возвращает значение бита под номером bitId в векторе key. Вторая возвращает позицию самого левого бита, который отличается у двух векторов. Реализация вектора в целом тривиальна, и также не нуждается в пояснениях.

Класс	Значение
class TString	вектор, используется для представления ключей
struct TNode	узел патриции, хранит ключ и значение
struct TPatricia	сама патриция. Поле только одно, это хэдер.
Функция	Значение
bool BitGet(...)	возвращает значение искомого бита в векторе
int32_t LeftMost(...)	возвращает номер первого слева отличающегося у двух векторов бита
int Save(...)	сохраняет данные о хэдере, обрабатывает возможные ошибки в открытии файла и запускает рекурсивное сохранение потомков хэдера
int RecursiveSave(...)	рекурсивное сохранение
int Load(...)	проверяет файл сохранения на целостность, последовательно считывает оттуда данные об узлах, и добавляет их в дерево
int main()	основная функция, одновременно реализующая пользовательский интерфейс

Дневник отладки

При создании этой таблицы была использована история посылок. К сожалению, мне доступны только последние 15, но их было значительно больше

Время	Проблема	Описание
2020/06/07 17:05:15	Превышено реальное время работы	Из-за неэффективности ввода программа работала слишком медленно
2020/06/08 16:00:25	Превышено реальное время работы	Отказался от посимвольного считывания данных, но он остался при преобразовании строки в вектор
2020/06/08 18:19:59	Работает	Убрал остатки посимвольного недоразумения, всё заработало.

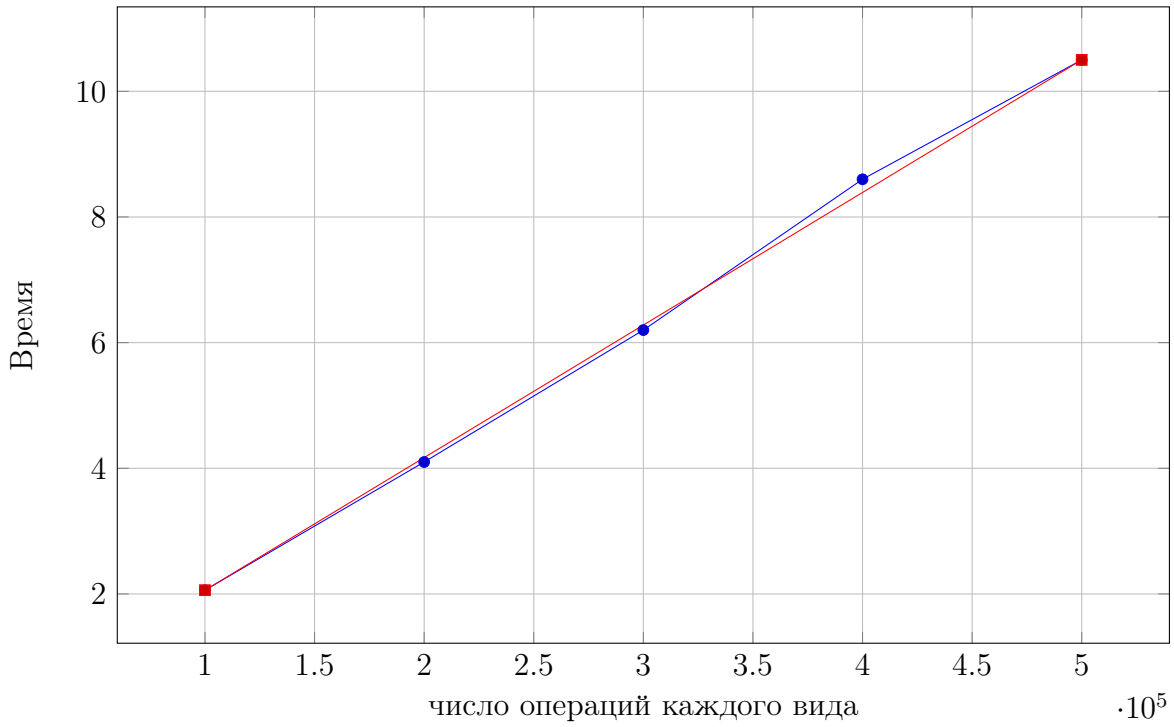
Тест производительности

Для генерации тестов использовалась следующая программа:

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>

int main(int argc, char **argv) {
    bool flag;
    size_t numLines = std::atoi(argv[1]);
    size_t numDels = std::atoi(argv[2]);
    size_t numSearches = std::atoi(argv[3]);
    std::string name = argv[4];
    std::string path = argv[5];
    std::ofstream os(name);
    srand(time(0));
    for (int i = 0; i < numLines; i++) {
        os << "+ ";
        for (size_t j = 0; j < rand() % 20 + 1; j++) {
            os << static_cast<char>(rand() % 20 + 97);
        }
        os << ' ' << rand();
        os << "\n";
    }
    for (int i = 0; i < numSearches; i++) {
        for (size_t j = 0; j < rand() % 20 + 1; j++) {
```

```
            os << static_cast<char>(rand() % 20 + 97);
        }
        os << "\n";
    }
    os << "! Save " << path << "\n";
    for (int i = 0; i < numDels; i++) {
        os << "- ";
        for (size_t j = 0; j < rand() % 20 + 1; j++) {
            os << static_cast<char>(rand() % 20 + 97);
        }
        os << "\n";
    }
    os << "! Load " << path << "\n";
    return 0;
}
```



Недочёты

Основной недочёт - это функции Save/Load. Их наверняка можно осделать эффективнее, что я оставлю для следующей лабораторной.

Выводы

Данная работа научила меня тому, что для написания эффективной программы стоит избегать посимвольных операций с текстом практически всегда, когда это в принципе возможно.