Лабораторная работа № 2 по курсу дискретного анализа: словарь

Выполнил студент группы М8О-208Б-17 МАИ *Милько Павел.*

Условие

- **Постановка задачи:**

  Необходимо создать программную библиотеку, реализующую указанную структуру данных, на основе которой разработать программу-словарь. В словаре каждому ключу, представляющему из себя регистронезависимую последовательность букв английского алфавита длиной не более 256 символов, поставлен в соответствие некоторый номер, от 0 до $2^{64} - 1$. Разным словам может быть поставлен в соответствие один и тот же номер.

- **Вариант дерева:**

  PATRICIA

- **Вариант ключа:**

  Регистронезависимая последовательность букв английского алфавита длиной не более 256 символов.

- **Вариант значения:**

  Числа от 0 до $2^{64} - 1$.

Метод решения

Нагруженное дерево (или Trie; Patricia - разновидность Trie-дерева) – структура данных реализующая интерфейс ассоциативного массива, то есть позволяющая хранить пары “ключ–значение”. В большинстве случаев ключами выступают строки, однако в качестве ключей можно использовать любые типы данных, представимые как последовательность байт. (то есть абсолютно любые)

В узлах Trie хранятся односимвольные метки, а ключом, который соответствует некоему узлу является путь от корня дерева до этого узла. Корень дерева, очевидно, соответствует пустому ключу.

Сжатое префиксное дерево PATRICIA является довольно быстрым и эффективным по памяти методом реализации словаря. Применение этой модели дерева может существенно увеличить продуктивность поиска и внесения элементов в словарь. Также есть не мало алгоритмов, которые построены на принципе дерева Patricia, например алгоритм поиска подстроки “Ахо — Корасик”.

Существует 2 основных типа оптимизации нагруженного дерева:

1. Сжатое нагруженное дерево получается из обычного удалением промежуточных узлов, которые ведут к единственному не промежуточному узлу. Например, цепочка промежуточных узлов с метками a, b, c заменяется на один узел с меткой abc.

2. PATRICIA нагруженное дерево получается из сжатого (или обычного) удалением промежуточных узлов, которые имеют одного ребенка.

Рассмотрим основные операции, связанные с префиксным деревом типа Patricia:

Операция поиска строки в префиксном дереве.

Движемся от корня дерева. Если корень пустой, то поиск неудачный. Иначе, сравниваем ключ в узле с текущей строкой. Для этого воспользуемся функцией, которая вычисляет длину наибольшего общего префикса двух строк заданной длины.

В случае поиска возможны три случая:

1. общий префикс может быть пустым, тогда надо рекурсивно продолжить поиск в младшей сестре данного узла, т.е. перейти по ссылке $right$;

2. общий префикс равен искомой строке x — поиск успешный, узел найден (тут мы существенно используем тот факт, что конец строки за счёт наличия в нем терминального символа может быть найден только в листе дерева);

3. общий префикс совпадает с ключом, но не совпадает с x — переходим рекурсивно по ссылке $left$ к старшему дочернему узлу, передавая ему для поиска строку x без найденного префикса.

Если общий префикс есть, но не совпадает с ключом, то поиск также является неудачным.

Операция вставки новой строки в префиксное дерево.

Вставка нового ключа (как и в двоичных деревьях поиска) очень похожа на поиск ключа. Естественно с несколькими отличиями. Во-первых, в случае пустого дерева нужно создать узел с заданным ключом и вернуть указатель на этот узел. Во-вторых, если длина общего префикса текущего ключа и текущей строки x больше нуля, но меньше длины ключа (второй случай неудачного поиска), то надо разбить текущий узел на два, оставив в родительском узле найденный префикс, и поместив в дочерний узел p оставшуюся часть ключа. После разделения нужно продолжить процесс вставки в узле p строки x без найденного префикса.

Удаление ключа из префиксного дерева.

Как обычно, удаление ключа — самая сложная операция. Хотя в случае префиксного дерева все выглядит не столь страшно. Дело в том, что при удалении ключа удаляется всего один листовой узел, соответствующий суффиксу некоторому удаляемого ключа. Сначала мы находим этот узел, если поиск успешный, то мы его удаляем и возвращаем указатель на младшего брата.

В принципе, на этом процесс удаления можно было бы и закончить, однако возникает небольшая проблема — после удаления узла в дереве может образоваться цепочка из двух узлов t и p, в которой у первого узла t имеется единственный дочерний узел p. Следовательно, если мы хотим держать дерево в сжатой форме, то нужно соединить эти два узла в один, произведя операцию слияния.

Описание программы

По условию задачи было ясно, что красиво такую программу написать в одном файле не получится и придется её разбить на несколько основных файлов:

1. main.cpp (содержит основной метод *main* и функцию *ToLower* – перевод строки в нижний регистр)
2. TTree.hpp и TTree.cpp (описание и реализация класса *TTree*)
3. StackContainer.hpp (описание и реализация класса *TStack* и *TStackData*)

Для удобства и автоматизации сборки был написан *Makefile*:

```
.PHONY : all, clean
TARGET = lab02
SRC    = $(wildcard *.cpp)
HDR    = $(wildcard *.hpp *.h)
CXX    = g++
CFLAGS = -pedantic -Wall -Wextra -Wpedantic -Wno-sign-compare \
         -Wno-long-long -O3

all: $(TARGET)

clean:
	rm -f $(TARGET) $(SRC:%.cpp=%.o)

$(TARGET): $(SRC:%.cpp=%.o)
	$(CXX) $(CFLAGS) -o $@ $^ -lm

main.o: main.cpp $(HDR)
	$(CXX) $(CFLAGS) -c main.cpp
```

```
%.o: %.cpp %.hpp
	$(CXX) $(CFLAGS) -c -o $@ $<
```

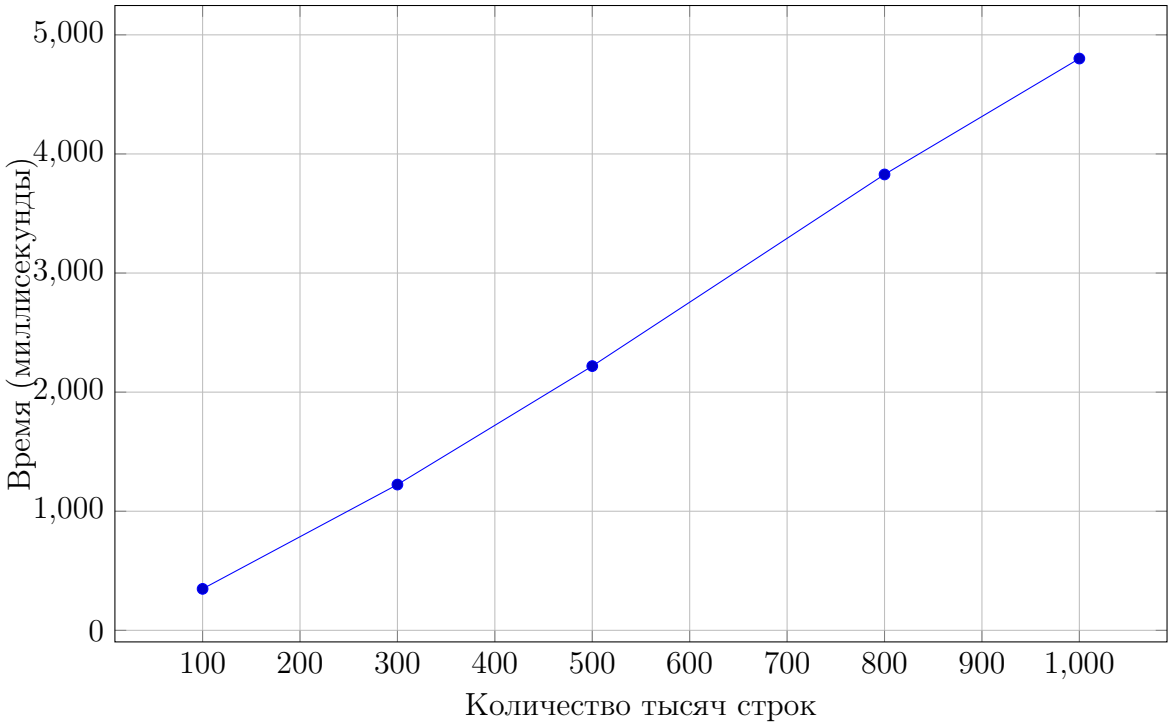
Генератор тестов:

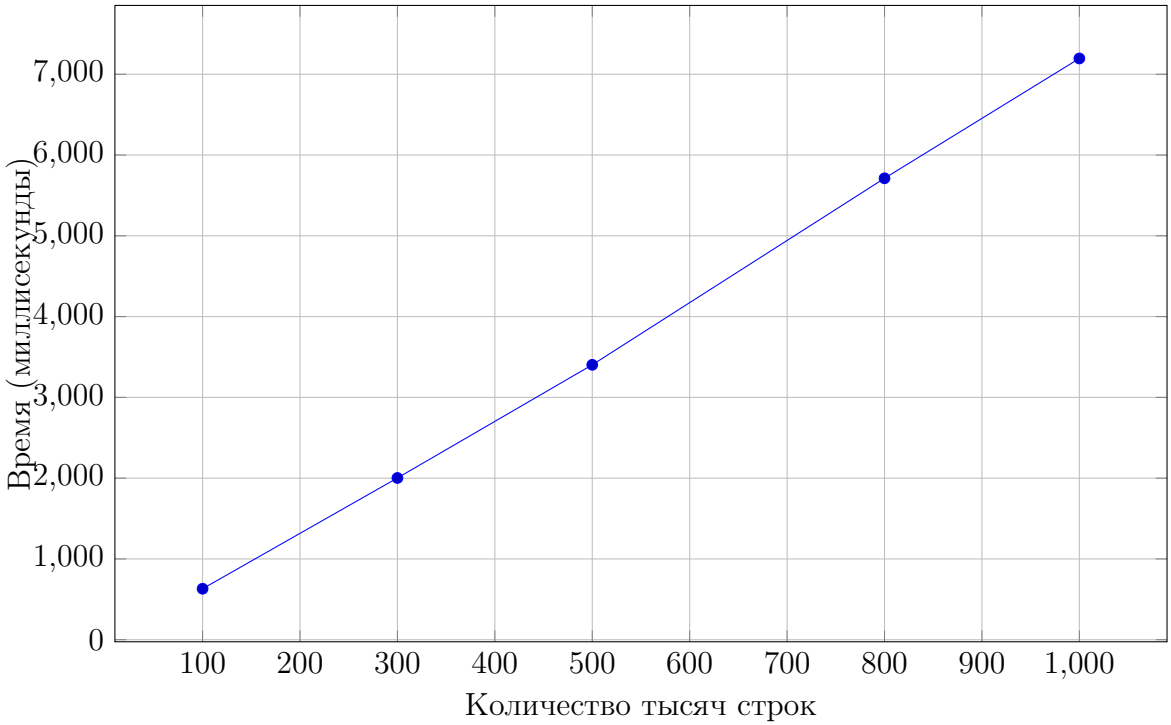
```
#!/usr/bin/python
import sys
from random import choice, randint
from string import ascii_uppercase

def get_random_key():
    return ''.join(choice(ascii_uppercase) for i in range(randint(1, 20)))

def main():
    if len(sys.argv) != 2:
        print("Usage: {0} <size of test>".format(
            sys.argv[0]))
        sys.exit(1)
    actions = [*["+"]*10, *["-"]*10, "?", "!"]
    acts_file = ["Load test", "Save test"]
    keys = {}
    test_file_name = "tests/"+sys.argv[1]
    with open("{0}.t".format(test_file_name), 'w') as output_file:
        #Для каждого файла генерируем от 1 до 100 тестов.
        for _ in range(int(sys.argv[1])):
            action = choice(actions)
            if action == "+":
                key = get_random_key()
                value = randint(1, 2**64-1)
                output_file.write("+ {0} {1}\n".format(key, value))
            elif action == "?":
                search_exist_element = choice([True, False])
                key = choice([key for key in keys.keys()])\
                 if search_exist_element and len(
                    keys.keys()) > 0 else get_random_key()
                output_file.write("{0}\n".format(key))
            elif action == "-":
                key = get_random_key()
                output_file.write("- {0}\n".format(key))
            elif action == "!":
                act_file = choice(acts_file)
                if act_file == "Save test":
```

```
                output_file.write(
                "{0}{1}\n".format(action, act_file))
        output_file.close()

if __name__ == "__main__":
    sys.exit(main())
```

Дневник отладки

Время	Коммит	Описание
13 11:07	init	Начало работы, есть только шаблоны файлов и функций
13 12:47	побитовое &	Был прочитан раздел в Седжавике, Кормане и Кнуте, посвящённый префиксным деревьям и патриции в частности
13 15:53	Sedgewik PATRICIA	Изменение структуры ключа, замена методов, на соответствующие алгоритмам из Сеждвика
13 17:53	maybe insert; broken destructor	Первая попытка сделать вставку (неудачная) и ужасные утечки памяти
13 20:40	add TTree::Show	Создан методт вывода содержимого дерева на экран, но при нерабочей вставке он оказался бесполезен
24 15:01	todo bit	Снова попытки вставки, выделение различающегося бита в двух строках
24 16:44	TTree!!!!!	Решил всё таки сделать работающий деструктор, но из за неправильной функции вставки он оказался невостребован
24 22:02	todo::again Insert	Отчаянная попытка сделать вставку (и последняя)
26 14:26	Trie	Нашёл более простую в реализации версию патриции, добавил вывод в файл с помощью стека
31 11:08	INSERT	Окончательно сделал вставку, в последствии убирал из нёё ненужные строки и выносил повторяющиеся действия в отдельные функции
2 23:10	FIle	Некорректный вывод в файл, ругался sanitizer, не все символы из ключа
3 02:45	OK	Чекер прислал OK, далее я убирал костыли и вносил косметические изменения





Выводы

В целом написание патриции принесло только боль, ненависть к окружающему миру и многодневную бессонницу из за нахождения на грани нескольких дедлайнов

одновременнно