Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет: “Информационные технологии и прикладная математика”
Кафедра: 806 “Вычислительная математика и программирование”

Лабораторная работа №2

по курсу “Дискретный анализ”

Студент	Полей-Добронравова А.В.
Группа	М8О-307Б-18
Преподаватель	Д.Е. Ильвохин
Дата	
Оценка	

Москва, 2021

Задача: Необходимо создать программную библиотеку, реализующую указанную структуру данных, на основе которой разработать программу-словарь.
Вариант структуры данных: PATRICIA.
Вариант ключа: Регистронезависимую последовательность букв английского алфавита длиной не более 256 символов.
Вариант значения: Некоторый номер, от 0 до 2^{64} - 1. Разным словам может быть поставлен в соответствие один и тот же номер.
Формат входных данных:
Входной файл со строками, содержащими команды следующего формата.

+ word 34 — добавить слово «word» с номером 34 в словарь. Программа должна вывести строку «OK», если операция прошла успешно, «Exist», если слово уже находится в словаре.
- word — удалить слово «word» из словаря. Программа должна вывести «OK», если слово существовало и было удалено, «NoSuchWord», если слово в словаре не было найдено.
word — найти в словаре слово «word». Программа должна вывести «OK: 34», если слово было найдено; число, которое следует за «OK:» — номер, присвоенный слову при добавлении. В случае, если слово в словаре не было обнаружено, нужно вывести строку «NoSuchWord».
! Save /path/to/file — сохранить словарь в бинарном компактном представлении на диск в файл, указанный парамером команды. В случае успеха, программа должна вывести «OK», в случае неудачи выполнения операции, программа должна вывести описание ошибки (см. ниже).
! Load /path/to/file — загрузить словарь из файла. Предполагается, что файл был ранее подготовлен при помощи команды Save. В случае успеха, программа должна вывести строку «OK», а загруженный словарь должен заменить текущий (с которым происходит работа); в случае неуспеха, должна быть выведена диагностика, а рабочий словарь должен остаться без изменений. Кроме системных ошибок, программа должна корректно обрабатывать случаи несовпадения формата указанного файла и представления данных словаря во внешнем файле.

Для всех операций, в случае возникновения системной ошибки (нехватка памяти, отсутствие прав записи и т.п.), программа должна вывести строку, начинающуюся с «ERROR:» и описывающую на английском языке возникшую ошибку.

1 Описание

Patricia-trie является улучшением trie-дерева. В начале кратко опишу идею структуры данных trie.
Trie - бинарное дерево, в котором узел ссылается на левое поддерево, где ключи(в двоичном представлении) имеют в следующем разряде цифру 0, правое поддерево - цифру 1 в следующем разряде. Есть два типа узлов: 1) узел, хранящий номер разряда для проверки
2) узел, хранящий сам ключ

Имеется несколько недостатков, решаемых с помощью Patricia. Во-первых, trie однонаправленное, что вынуждает создавать дополнительные узлы. Во-вторых два различных типа узлов усложняет работу с trie.

В Patricia индекс разряда помещается в поле каждого узла. Существуют обратные указатели, для избежания дублирования узлов.
Описание идей функций для работы с Patricia:

1. Поиск

Для нахождения ключа theKey используем биты слева-направо, спускаясь вниз по дереву. Движение вниз, пока не встретится связь, указывающая вверх. Если ключ в узле, указанный первой направленной вверх связью равен theKey, успех. Иначе неуспех.

2. Вставка

Функция поиска приводит к уникальному ключу в дереве, который должен отличаться от вставляемого. Определяем самый левый разряд отличия между найденным и вставляемым. Далее при помощи вспомогательной функции insertR перемещаемся вниз по дереву и вставляем новый узел.
В функции insertR два случая. Новый узел может замещать внутреннюю связь(если ключ поиска отличается от ключа, найденного в позиции разряда, который был пропущен) или внешнюю связь(если разряд, отличающий искомый от найденного, не потребовался для различения найденного ключа от всех других)

3. Удаление

Два случая:

1) Удаляемый узел имеет указатель на себя. Если это голова дерева, то удалить дерево. Если это не голова, указатель с родителя на наш узел заменяем на указатель на нашего сына.
2) Удаляемый узел не имеет указатель на себя. Следовательно имеется другой узел q, указывающий вверх на удаляемый. Нужно заменить этот указатель на родителя удаляемого узла.

2 Исходный код

```
#include <cstring>

#include <iostream>

class TPatricaia {

    public:

        TPatricaia(): root(nullptr){}

        unsigned long long Find(char*
key) {

            if(root) {

                Node* tmp;

                if (root->left ==
root) {

                    tmp = root;

                } else {

                    tmp =
root->left->Find(key);

                }

                if (strcmp(tmp->key,
key) == 0) {

                    return tmp->value;

                }

            }

            throw "No Elem";

        }

        bool Add(char* key, unsigned
long long value) {

            if(root) {

                Node* tmp;

                if (root->left ==
root) {

                    tmp = root;

                } else {

                    tmp =
root->left->Find(key);

                }



                int NewBit =
getFirstDiffBit(key, tmp->key);

                //std::cout << NewBit
<< '\n';



                if (NewBit == -1) {

                    return false;

                }

                Node** child;

                if (root->left == root
|| root->left->Nbit > NewBit) {

                    tmp = root;
```

```
                    child =
&(tmp->left);

                } else {

                    tmp =
root->left->Find(key, NewBit);

                    if (getNBit(key,
tmp->Nbit)) {

                        child =
&(tmp->left);

                    } else {

                        child =
&(tmp->right);

                    }

                }



                Node* NewNode = new
Node(key, value, NewBit);

                NewNode->parantPtr =
child;

                if
(getNBit(NewNode->key, NewNode->Nbit))
{

                    NewNode->left =
NewNode;

                    NewNode->right =
*child;

                    if ((*child)->Nbit
> NewNode->Nbit) {


(*child)->parantPtr =
&(NewNode->right);

                    }

                } else {

                    NewNode->right =
NewNode;

                    NewNode->left =
*child;

                    if ((*child)->Nbit
> NewNode->Nbit) {


(*child)->parantPtr =
&(NewNode->left);

                    }

                }

                *(NewNode->parantPtr)
= NewNode;

                return true;

            } else {

                root = new Node(key,
value, -1);

                root->right = nullptr;

                root->left = root;

                return true;

            }

        }

        bool Del(char* key) {

            if(root) {

                Node* tmp;

                if (root->left ==
root) {

                    if
(strcmp(root->key, key) == 0) {

                        delete root;

                        root =
nullptr;

                        return true;

                    } else {

                        return false;

                    }

                } else {
```

```
                    tmp =
root->left->Find(key,  __INT_MAX__ -
1);

                    Node** child;



                    if (getNBit(key,
tmp->Nbit)) {

                        child =
&(tmp->left);

                    } else {

                        child =
&(tmp->right);

                    }

                    if
(strcmp((*child)->key, key) != 0) {

                        return false;

                    }

                    if ((*child) !=
root) {

                        if ((*child)
== tmp) {

                            Node*
NormalChild;

                            if
(getNBit(key, tmp->Nbit)) {


NormalChild = tmp->right;

                            } else {


NormalChild = tmp->left;

                            }

                            if
(NormalChild->Nbit > tmp->Nbit) {


NormalChild->parantPtr =
tmp->parantPtr;

                            }


*(tmp->parantPtr) = NormalChild;

                            tmp->left
= nullptr;

                            tmp->right
= nullptr;

                            delete
tmp;

                            return
true;


                        }

                    }



                    //Case 2

strcpy((*child)->key, tmp->key);

                    (*child)->value =
tmp->value;


                    Node* prev =
tmp->Find(tmp->key,  __INT_MAX__ - 1);

                    if
(getNBit(tmp->key, prev->Nbit)) {

                        prev->left =
*child;

                    } else {

                        prev->right =
*child;

                    }


                    Node* NormalChild;

                    if (getNBit(key,
tmp->Nbit)) {
```

```
                        NormalChild =
tmp->right;
                    } else {
                        NormalChild =
tmp->left;
                    }


                    if
(NormalChild->Nbit > tmp->Nbit) {

NormalChild->parantPtr =
tmp->parantPtr;
                    }
                    *(tmp->parantPtr)
= NormalChild;
                    tmp->left =
nullptr;
                    tmp->right =
nullptr;
                    delete tmp;
                    return true;
                }


            } else {
                return false;
            }
        }


        bool Save(char* path) {
            FILE* stor = fopen(path,
"wb");
            if (!stor) {
                return false;
            }
            if (root) {
                root->Save(stor);
            }
            int s = 0;
            fwrite(&s, sizeof(int), 1,
stor);
            fclose(stor);
            return true;
        }
        bool Load(char* path) {
            FILE* stor = fopen(path,
"rb");
            if (!stor) {
                return false;
            }
            if(root){
                delete root;
                root = nullptr;
            }
            int flag;
            char key[257] = {0};
            unsigned long long value;
            size_t res;
            res = fread(&flag,
sizeof(int), 1, stor);
            if (res != 1) {
              return false;
            }
            while (flag) {
                res = fread(key,
sizeof(char) * 257, 1, stor);
                if (res != 1) {
```

```
                    return false;

                }

                res = fread(&value,
sizeof(unsigned long long), 1, stor);

                if (res != 1) {

                    return false;

                }

                Add(key, value);

                res = fread(&flag,
sizeof(int), 1, stor);

                if (res != 1) {

                    return false;

                }

            }

            return true;

        }



        ~TPatricaia(){

            if(root){

                delete root;

            }

        }



    private:

        static int
getFirstDiffBit(char* a, char* b) {

            int out = 0;

            int i = 0;

            while(a[i] != '\0' && b[i]
!= '\0') {

                if (a[i] == b[i]) {

                    out += 8;

                } else {

                    for (int k = 0; k
< 8; k++) {

                        if ((a[i] &
(1u<<k)) == (b[i] & (1u<<k))) {

                            out += 1;

                        } else {

                            return
out;

                        }

                    }

                }

                i++;

            }

            for (int k = 0; k < 8;
k++) {

                if ((a[i] & (1u<<k))
== (b[i] & (1u<<k))) {

                    out += 1;

                } else {

                    return out;

                }

            }

            return -1;

        }

        static int getNBit(char*
InKey, int NBit) {

            return
(bool)(InKey[(int)(NBit/8)] & (1u <<
(NBit % 8)));

        }

        struct Node {

            char key[257];

            unsigned long long value;
```

```
            int Nbit;

            Node* left;

            Node* right;

            Node** parantPtr;



            Node(char* InKey,unsigned
long long value,  int NBit): key{0},
value(value),  Nbit(NBit){

                strcpy(key, InKey);

            }



            Node* Find(char* InKey,
int NewBit = __INT_MAX__) {

                if (getNBit(InKey,
Nbit)) {

                    if (left->Nbit >
NewBit) {

                        return this;

                    }

                    if (left->Nbit <=
Nbit) {

                        if (NewBit ==
__INT_MAX__) {

                            return
left;

                        } else {

                            return
this;

                        }

                    }

                    return
left->Find(InKey, NewBit);

                } else {

                    if (right->Nbit >
NewBit) {

                        return this;

                    }

                    if (right->Nbit <=
Nbit) {

                        if (NewBit ==
__INT_MAX__) {

                            return
right;

                        } else {

                            return
this;

                        }

                    }

                    return
right->Find(InKey, NewBit);

                }

            }

            int Save(FILE* stor) {

                int s = 1;

                fwrite(&s,
sizeof(int), 1, stor);

                fwrite(key,
sizeof(char) * 257, 1, stor);

                fwrite(&value,
sizeof(unsigned long long), 1, stor);

                if (left) {

                    if (left->Nbit >
Nbit) {


left->Save(stor);

                    }

                }

                if (right) {

                    if (right->Nbit >
Nbit) {
```

```
right->Save(stor);
                    }
                }
                return true;
            }
            ~Node(){
                if (left) {
                    if (left->Nbit >
Nbit) {
                        delete left;
                    }
                }
                if (right) {
                    if (right->Nbit >
Nbit) {
                        delete right;
                    }
                }
            }
        };

        Node* root;
};

void tolow(char* a) {
    for (int i = 0; a[i] != '\0'; i++)
{
        if (a[i] >= 'A' && a[i] <=
'Z') {
            a[i] = tolower(a[i]);
        }
    }
}

int main() {
    std::ios::sync_with_stdio(false);
    char comm[50];
    char key[257] = {'\0'};
    unsigned long long value;

    TPatricaia stor;

    while(std::cin >> comm) {
        if (comm[0] == '+') {
            std::cin >> key >> value;
            tolow(key);
            if(stor.Add(key,value)) {
                puts("OK");
            } else {
                puts("Exist");
            }
        } else if (comm[0] == '-') {
            std::cin >> key;
            tolow(key);
            if(stor.Del(key)) {
                puts("OK");
            } else {
                puts("NoSuchWord");
            }
        } else if (comm[0] == '!') {
```

```
            std::cin >> comm >> key;
            if (comm[0] == 'S') {
                if(stor.Save(key)) {
                    puts("OK");
                } else {
                    puts("ERROR");
                }
            } else if (comm[0] == 'L')
{
                if(stor.Load(key)) {
                    puts("OK");
                } else {
                    puts("ERROR");
                }
            }
        } else {
            try {
                tolow(comm);
                value =
stor.Find(comm);
                printf("OK: %llu\n",
value);
            } catch (...) {
                puts("NoSuchWord");
            }
        }
    }
    return 0;
}
```

3 Примеры прохождения тестов

test.txt

“aps
+ aps 125
+ aps 125
aps
+ heey 15
+ sorry 16
+ privet 13
aps
heey
sorry
- heey
aps
heey
sorry
privet
porp”

```
 MacBook-Pro-Amelia:2lab amelia$ ./da2 < test.txt
NoSuchWord
OK
Exist
OK: 125
OK
OK
OK
OK: 125
OK: 15
OK: 16
OK
OK: 125
NoSuchWord
OK: 16
OK: 13
NoSuchWord
```

test2.txt

“+ first 1
+ trin 13
+ three 3
+ nine 9
+ eight 8
+ twelfe 12
first
three
twelfe
- trin
twelfe
trin”

```
MacBook-Pro-Amelia:2lab amelia$ ./da2 < test2.txt
OK
OK
OK
OK
OK
OK
OK: 1
OK: 3
OK: 12
OK
OK: 12
NoSuchWord
```

test3.txt пустой файл

```
MacBook-Pro-Amelia:2lab amelia$ ./da2 < test3.txt
MacBook-Pro-Amelia:2lab amelia$
```

4 Выводы

В данной лабораторной нужно было только реализовать определенную структуру данных, опираясь на конкретные готовые описания ее строения и функций. Сама реализация не дала демонстрацию эффективности Patricia, но если провести отдельный анализ, то можно выяснить следующие вещи.

Вставка или поиск случайного случайного ключа, построенного из N случайных строк разрядов, требует приблизительно lg(N) сравнений разрядов в среднем и приблизительно 2 lg(N) сравнений разрядов в худшем случае. *Затраты на поиск не возрастают с увеличением длины ключа.*

Также Patricia особенно эффективна в задачах с переменной длиной ключей.