Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект
по курсу «Операционные системы»
III Семестр

Визуализация сжатого префиксного дерева

Студент:	Короткевич Л. В.
Группа:	М80-208Б-19
Преподаватель:	Миронов Е.С
Оценка:	
Дата:	

Москва 2020

1. Цель курсового проекта.

Приобретение практических навыков в использовании знаний, полученных в течении курса.

Проведение исследования в выбранной предметной области.

2. Задание.

Визуализация структуры данных сжатое префиксное дерево.

3. Введение.

Данную программу я описал для того, чтобы упростить жизнь себе и другим студентам, познающим сложные структуры данных в роде Patricia Trie и ей подобных.

В момент изучения материала на тему сжатых префиксных деревьев мне крайне не хватало визуальных примеров и «картинки перед глазами», которая демонстрировала бы, какие изменения претерпевает дерево при удалении/вставке в неё различных элементов.

Для разрешения данной проблемы было принято решение создать «интерактивную» программу, которая позволяла бы отслеживать изменения дерева по мере её редактирования.

4. *Ликбез: его величество* Patricia-Trie.

Patricia Trie - специальная структура данных, необходимая для хранения ключей (чаще всего - слов). Узлы дерева содержат в себе пару ссылок на узлы (или на себя, или на «младшие», а то и «старшие» узлы), ключ, значение и номер бита, который мы будем сравнивать при проходе через этот узел.

Как не трудно догадаться, все операции основаны на сравнении определённых битов, так что временная сложность операций поиска, добавления и удаления элемента из дерева оценивается как O(k), где k — длина обрабатываемого элемента. Очевидно, что время работы не зависит от количества элементов в дереве.

Данная реализация имеет одну закономерность - узлы дерева отсортированы (по мере прохождения вглубь) по возрастанию по

номеру бита, который они хранят. Это позволит нам определять, является ли ссылка указателем “наверх” без использования специальных меток.

Что отличает Патрицию от её предшественников (Trie, Prefix Trie) — в ней не происходит одностороннего ветвления, что обусловлено наличием того самого индекса — номера бита, необходимого для сравнения ключей.

5. Техническая сторона вопроса.

5.1. Реализация Patricia-Trie.

Структуру узла, конструкторы и деструкторы дерева я описал в Node.cpp, а структуру дерева со всеми необходимыми операции для работы с ним — в Trie.cpp. Дополнительные функции для работы со строками, битами и т.д. вынесены в Additional.hpp.

Отдельно стоит сказать о (де-)сериализации дерева. Сохранение дерева реализовано следующим образом: прежде чем записать информацию об узлах в поток, я нумерую их. Сделано это за тем, чтобы упростить себе жизнь: вместо рекурсивного вызова функции сериализации для дочерних узлов при сохранении какого-то из них, я буду писать в поток номера его детей (нумерация может быть любой; например, я их нумеровал последовательно при прямом обходе). Десериализация происходит аналогично сериализации, узлы считываются в том же порядке.

5.2. Создание статической библиотеки для работы с Patricia.

Для меня было крайне приятным открытие того факта, что пользователи Linux-систем могут спокойно обмениваться между собой исполняемыми файлами, запуская их друг у друга. Ещё больше я обрадовался, когда узнал о возможностях статической линковки: свою программу можно «связать» с какой-либо библиотекой N, а затем открывать её где угодно без необходимости иметь установленной на компьютер эту самую библиотеку N.

Таким образом было принято решение описать статическую библиотеку для работы с Patricia Trie.

Сие деяние было реализовано незамысловатым образом: описав саму структуру данных, раскидав по файлам объявление и описание методов для работы с Patricia, я выполнил следующий набор действий:

$ g++ -c Trie.cpp Node.cpp // компилирую программы дерева и узла в объектные файлы

$ ar rc libpatricia Node.o Trie.o // создание библиотеки из объектных файлов

$ g++ main.cpp -lpatricia -L./ -o main // компиляция основного файла: укажем каталог библиотеки и имя библиотеки

Таким образом я создал статическую библиотеку, а затем «связал» её с основной программой.

5.3. Визуализация дерева.

Наконец, описав весь необходимый для работы с Patricia набор методов, я принялся за реализацию визуального изображения оной. За сим я обратился в интернет для разрешения вопроса; после долгих поисков наткнулся на прекрасную утилиту для автоматической визуализации графов, заданных в виде описания на языке DOT.

По мере освоения работы с данной утилитой я принялся писать функционал программы, реализующий генерацию и описание файла source.dot, в котором будет описана структура Patricia. Решил этот вопрос незамысловатым образом: понадобилось всего две основных функции. Первая описывает каждый узел в отдельности, как он будет изображаться на картинке и его «лейблы». Вторая же описывает взаимосвязь между всеми узлами.

5.4. Многопроцессорность, замена образов процессов и их функционал.

В целях приобретения практических навыков было приятно решение описать следующую логистику «размногопроцессирования» программы: пусть в теле основной программы будет создано разветвление на два процесса, один из которых, очевидно, — дочерний. В нём я опишу создание файла source.dot с помощью

функционала дерева, наличествующего в статической библиотеке, созданной ранее.

Дальше — больше: пусть и этот дочерний процесс раздвоится. На этот раз я использую семейство функций exec (а именно - execlp), заменяющих текущий образ процесса новым образом процесса с двумя дополнительными аргументами при запуске: файл для сохранения изображения патриции и название утилиты, с помощью который будет происходить открытие изображения. Для этого я описал отдельную программу drawer.cpp, реализующую «прорисовку» изображения на основе файла source.dot с помощью функций из динамической библиотеки, подключаемой в программе во время её исполнения.

5.5. Динамическая библиотека draw, подключаемая во время исполнения.

Для закрепления пройденного материала я решил описать динамическую библиотеку для рисования и обозревания изображения Patricia. Более того, для усложнения задачи было принято решение: а) подключать библиотеку во время выполнения программы и б) написать всё на C++.

5.6. Работа со сторонними утилитами внутри программы.

До прохождения курса по ОС я не задумывался, что работа с командной оболочкой Linux внутри программы настолько проста. В реализации контрактов динамической библиотеки draw мне помог библиотечный вызов system(const char * string), вызывающий в свою очередь /bin/sh -c string.

Незамысловатым образом, согласно синтаксису утилиты Graphviz, я выполняю в командной оболочке действия, генерирующих изображение Patricia на основе файла source.dot.

Затем аналогичным образом я запускаю с помощью того же библиотечного вызова system указанный вторым аргументом при запуске программы worker обозреватель изображения, будь то: xdg-open, kolourpaint, feh или любая другая утилита.

6. Тестирование работоспособности.

Команды для работы с программой:

- **+ <word> <value>**: добавить слово word с ключом value
- **- <word>**: удалить слово word
- **<word>**: проверить наличие слова word в словаре
- **! Save <filename>**: сохранить структуру дерева в файл
- **! Load <filename>**: загрузить -/-/-
- **print <filename> <toolname>**: генерация изображения дерева filename.png, визуализация дерева с помощью утилиты toolname

6.1. Тестирование общей работоспособности словаря.

Test1:

! Save db0
- bb
! Load db0
ca
- bb

Res1:

OK
NoSuchWord
OK
NoSuchWord
NoSuchWord

Test2:

- ba
c
- ccc
+ a 1114405545
+ ccb 510116504
- abb
+ a 1628686999
c
bc
bcb

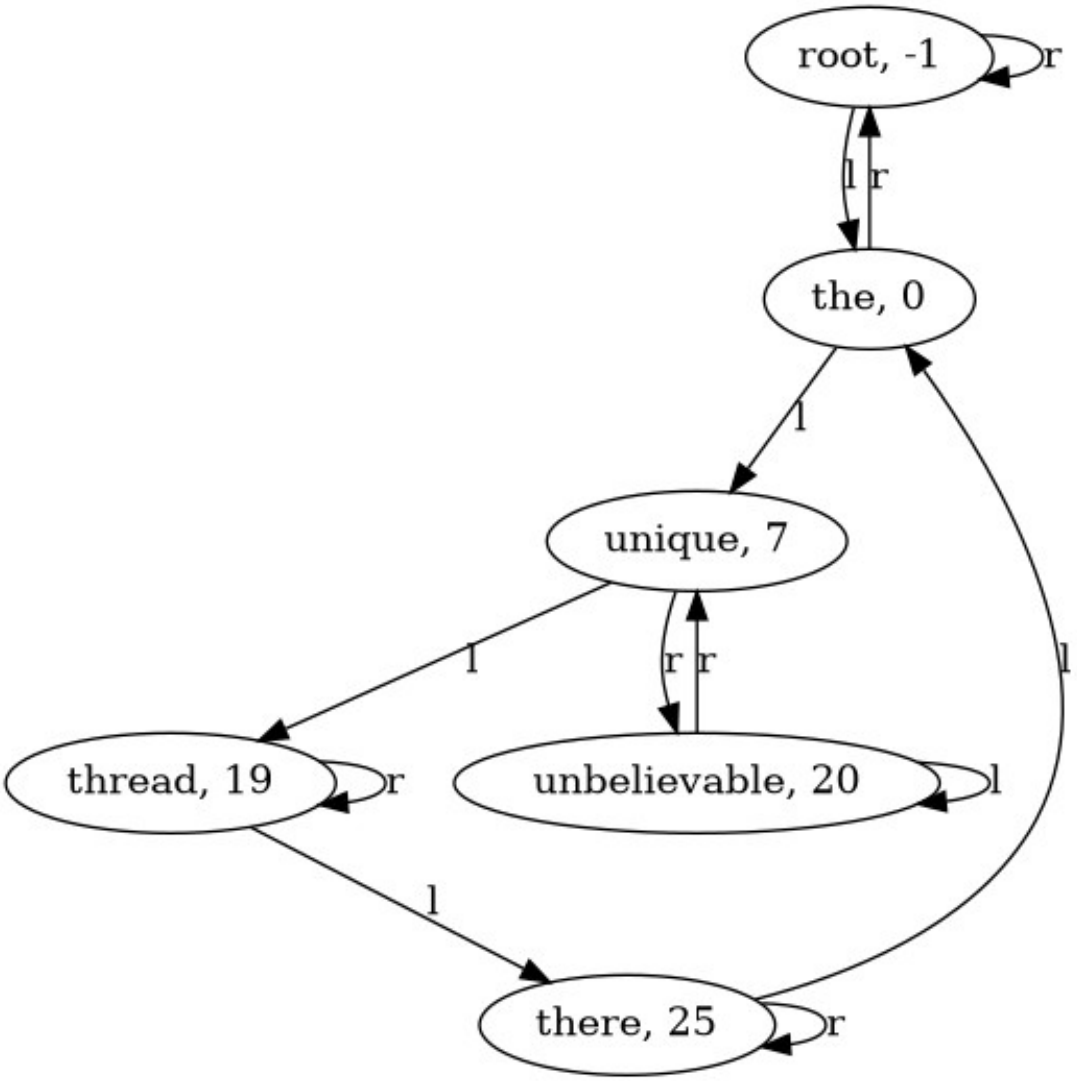
Res2:
NoSuchWord
NoSuchWord
NoSuchWord
OK
OK
NoSuchWord
Exist
NoSuchWord
NoSuchWord
NoSuchWord
Test3:
- ba
c
bca
- bb
! Save db1
+ caa 1827191341
+ bca 1875584253
c
+ abb 765090149
- ba
- c
ca
! Save db0
- bca
+ a 1294963798
Res3:
NoSuchWord
NoSuchWord
NoSuchWord
NoSuchWord
OK
OK

OK
NoSuchWord
OK
NoSuchWord
NoSuchWord
NoSuchWord
OK
OK

6.2. Примеры визуализации дерева.

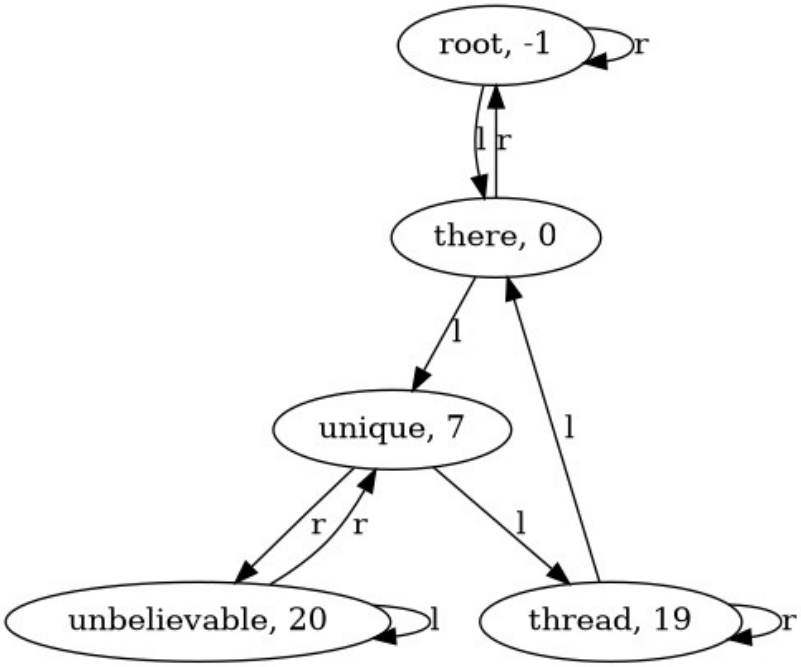
Во время исполнения программы пользователю достаточно ввести необходимую команду, как перед ним откроется изображение со структурой его дерева на текущий момент.

Примеры таких изображений:

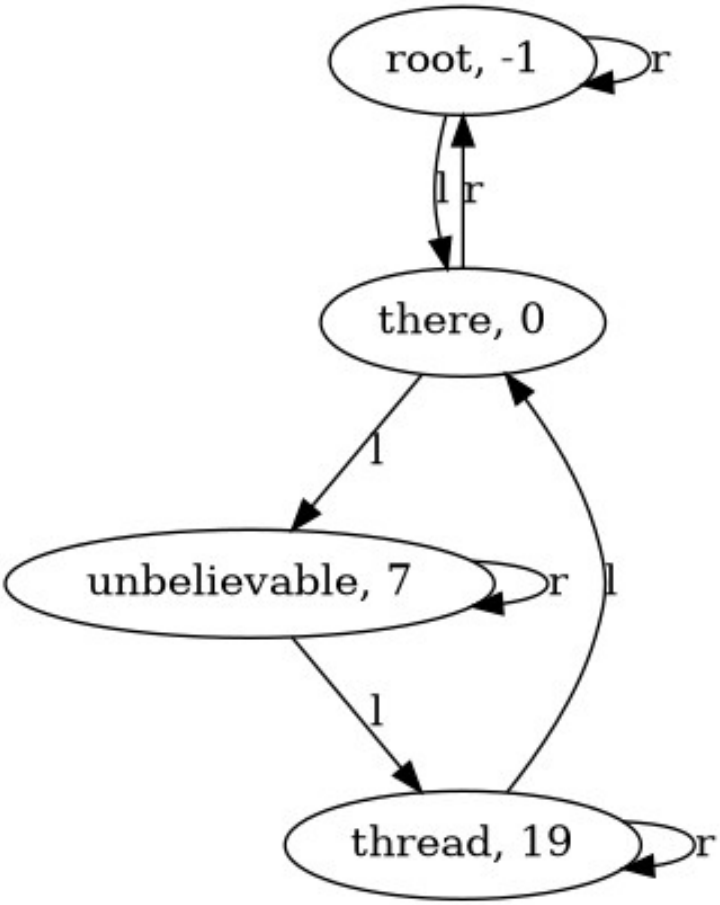


На этом же примере дерева продемонстируем его поэтапную деконструкцию.

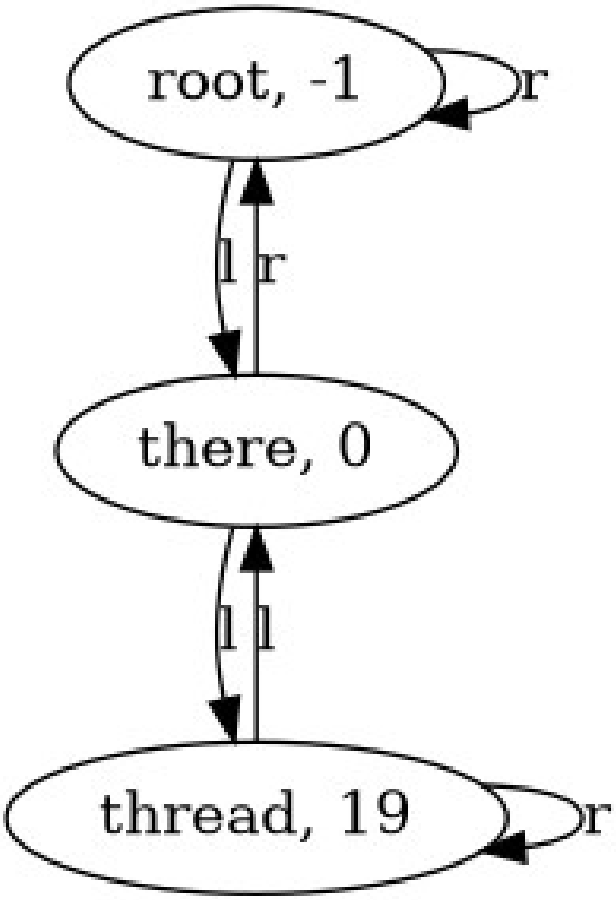
Шаг **№1:** удаление слова the:



Шаг **№2:** удаление слова unique:



Шаг **№3**: удаление слова unbelievable:



7. Тестирование производительности.

Сравнение производительности будет производится с красно чёрным деревом, представленном в стандартной библиотеке шаблонов C++ – контейнером map.

Тест №1: добавление 1000 элементов.

Тест №2: добавление 1000000 элементов.

Тест №3: добавление 1000 элементов, удаление каждого 4-го.

Тест №4: добавление 1000000 элементов, удаление каждого 4-го.

Тест №5: сначала добавление 1000 элементов, потом их удаление.

Тест №6: сначала добавление 1000000 элементов, потом их удаление.

Время исполнения представлено в микросекундах.

Test	Map, ms	PATRICIA, ms	Times faster
1	368	179	2,055865922
2	336192	178322	1,885308599
3	2238	725	3,086896552
4	512560	154529	3,31691786
5	2289	1195	1,915481172
6	431921	285703	1,511783215

Как оказалось, для хранения слов Patricia Trie оказалась много эффективней красно-чёрного дерева. Не удивительно, ведь сложность операций поиска/добавления/удаления в КЧД — O(log(n)) , где n — число элементов в дереве; в случае же Патриции, сложность этих же операций — O(k), где k — длина слова. Притом Патриция сравнивает лишь определённые биты слов во время их поиска, а КЧД же – все слово целиком. Нет так же затрат на балансировку Патриции в отличии от КЧД. Наблюдательно, что Патриция в результате вставки случайных слов сама по себе является почти идеально сбалансированным деревом, и тем более она таковой является, чем длиннее слова в неё вставляемые.

8. Листинг программы.

```
[leo@pc ver1_static_fork]$ cat PatFuncs/Trie.hpp
#pragma once

#include "Node.hpp"
#include <fstream>

struct TTrie {
    TNode *root;
    int size;

    TTrie();
    void DestructR(TNode *node);

    ~TTrie();

    TNode *Find(TKey *key);

    TNode *Insert(TKey *key, TValue value);
```

```
    void KVCopy(TNode *src, TNode *dest);

    bool Delete(TKey *k);

    void Save(std::ofstream &file);

    void enumerate(TNode *node, TNode **nodes, int &index);

    void Load(std::ifstream &file);

    void PrintDefinitions(TNode *node, std::ofstream &out);

    void PrintRelations(TNode *node, std::ofstream &out);
};
[leo@pc ver1_static_fork]$ cat PatFuncs/Trie.cpp
#include "Trie.hpp"
#include "Additional.hpp"

#include <cstring>
#include <fstream>
#include <iostream>

static inline std::string safeKey(const TNode *node)
{
    return node->key ? node->key : "root";
}

TTrie::TTrie()
{
    root = new TNode();
    size = 0;
}

void TTrie::DestructR(TNode *node)
{
    if (node->left->bit > node->bit)
        DestructR(node->left);
    if (node->right->bit > node->bit)
        DestructR(node->right);
    delete node;
}

TTrie::~TTrie()
{
    DestructR(root);
}

TNode *TTrie::Find(TKey *key)
{
    TNode *p = root;
    TNode *q = root->left;

    while (p->bit < q->bit) {
        p = q;
        q = (GetBit(key, q->bit) ? q->right : q->left);
```

```
    }

    if (!Equal(key, q->key))
        return 0;

    return q;
}

TNode *TTrie::Insert(TKey *key, TValue value)
{
    TNode *p = root;
    TNode *q = root->left;
    while (p->bit < q->bit) {
        p = q;
        q = (GetBit(key, q->bit) ? q->right : q->left);
    }

    if (Equal(key, q->key))
        return 0;

    int lBitPos = FirstDifBit(key, q->key);

    p = root;
    TNode *x = root->left;

    while (p->bit < x->bit && x->bit < lBitPos) {
        p = x;
        x = (GetBit(key, x->bit) ? x->right : x->left);
    }

    try {
        q = new TNode();
    }
    catch (const std::bad_alloc &e) {
        std::cout << "ERROR: fail to allocate the requested storage space\n";
        return 0;
    }

    q->Initialize(lBitPos, key, value,
                  (GetBit(key, lBitPos) ? x : q),
                  (GetBit(key, lBitPos) ? q : x));

    if (GetBit(key, p->bit))
        p->right = q;
    else
        p->left = q;

    size++;
    return q;
}

void TTrie::KVCopy(TNode *src, TNode *dest)
{
    if (strlen(dest->key) < strlen(src->key)) {
        delete[] dest->key;
```

```
        dest->key = new char[strlen(src->key) + 1];
    }
    strcpy(dest->key, src->key);

    dest->value = src->value;
}

bool TTrie::Delete(TKey *k)
{
    // прадед удаляемого узла pp, родитель p и сам сын t (сына предстоит
удалить)
    TNode *p, *t, *pp = 0;

    p = root;
    t = (p->left);

    // найдем pp, p и t
    while (p->bit < t->bit) {
        pp = p;
        p = t;
        t = (GetBit(k, t->bit) ? t->right : t->left);
    }

    // если ключа искомого-то и нет -- выходим
    if (!Equal(k, t->key))
        return false;

    TNode *x, *r;
    char *key;

    // если p == t, то у t есть селфпоинтер. в таком случае достаточно лишь
    // переподвесить к родителю (pp) "реальный" указатель t, который не
селфпоинтер
    if (p != t) {
        // иначе же, кладем ключ и знач. p в t, чтобы далее удалять именно p, а
не t
        KVCopy(p, t);

        key = p->key;
        r = p;
        x = (GetBit(key, p->bit) ? p->right : p->left);

        // ищем того, кто на p бекпоинтерит (будет лежать в r; а x, по сути,
будет в точности равняться p)
        while (r->bit < x->bit) {
            r = x;
            x = (GetBit(key, x->bit) ? x->right : x->left);
        }

        // и вместо бекпоинтера на p, будем бекпоинтерить на t
        if (GetBit(key, r->bit))
            r->right = t;
        else
            r->left = t;
    }
```

```
    // остается подвесить к родителю p (pp) "реальный" указатель p, который не
селфпоинтер
    TNode *ch = (GetBit(k, p->bit) ? p->left : p->right);
    if (GetBit(k, pp->bit))
        pp->right = ch;
    else
        pp->left = ch;

    // и беззаботно удалить p: больше на него никто не указывает, ведь мы
избавились
    // и от бекпоинтера на него, и от родительского (pp) указателей сверху
    delete p;

    size--;

    return true;
}

void TTrie::Save(std::ofstream &file)
{
    // подаем размер дерева
    file.write((const char *)&(size), sizeof(int));

    // пронумеровка узлов, инициализация массива указателей
    int index = 0;
    TNode **nodes;
    try {
        nodes = new TNode *[size + 1];
    }
    catch (const std::bad_alloc &e) {
        std::cout << "ERROR: fail to allocate the requested storage space\n";
        return;
    }
    enumerate(root, nodes, index);

    // теперь просто последовательно (как при обходе в enumerate)
    // подаем всю инфу об узлах, но вместо указателей left/right подаем
    // айди узлов (каковы они были при обходе в enumerate) left/right
    TNode *node;
    for (int i = 0; i < (size + 1); ++i) {
        node = nodes[i];
        file.write((const char *)&(node->value), sizeof(TValue));
        file.write((const char *)&(node->bit), sizeof(int));
        int len = node->key ? strlen(node->key) : 0;
        file.write((const char *)&(len), sizeof(int));
        file.write(node->key, sizeof(char) * len);
        file.write((const char *)&(node->left->id), sizeof(int));
        file.write((const char *)&(node->right->id), sizeof(int));
    }
    delete[] nodes;
}

void TTrie::enumerate(TNode *node, TNode **nodes, int &index)
{
```

```
    // важно, что index передается по ссылке: айди узлов не будут повторяться
    node->id = index;
    nodes[index] = node;
    ++index;
    if (node->left->bit > node->bit) {
        enumerate(node->left, nodes, index);
    }
    if (node->right->bit > node->bit) {
        enumerate(node->right, nodes, index);
    }
}

void TTrie::Load(std::ifstream &file)
{
    // считываем размер
    int n;
    file.read((char *)&n, sizeof(int));
    size = n;
    // если он нуль - выходим
    if (!size)
        return;

    TNode **nodes = new TNode *[size + 1];
    // рут уже инициализировался, когда мы пишем создали new Trie()
    // незачем этого делать повторно
    nodes[0] = root;
    for (int i = 1; i < (size + 1); ++i)
        // а вот прочие узлы надо инитнуть
        nodes[i] = new TNode();

    // поля узлов, которые нам предстоит считывать
    int bit;
    int len;
    TKey *key = 0;
    TValue value;
    int idLeft, idRight;

    for (int i = 0; i < (size + 1); ++i) {
        file.read((char *)&(value), sizeof(TValue));
        file.read((char *)&(bit), sizeof(int));
        file.read((char *)&(len), sizeof(int));
        if (len) {
            key = new char[len + 1];
            key[len] = 0;
        }
        file.read(key, len);
        // поскольку считываем в том же порядке, что и писали в Load-е
        // айди узлов-сыновей будут сохранять свой порядок, и дерево соберется
таким же
        file.read((char *)&(idLeft), sizeof(int));
        file.read((char *)&(idRight), sizeof(int));
        nodes[i]->Initialize(bit, key, value, nodes[idLeft], nodes[idRight]);
        delete[] key;
    }
```

```
    delete[] nodes;

    return;
}

void TTrie::PrintDefinitions(TNode *node, std::ofstream &out)
{
    out << ' ' << safeKey(node) << "[label=\"" << safeKey(node) << ", " << node-
>bit << "\"];\n";
    if (node->left->bit > node->bit)
        PrintDefinitions(node->left, out);
    if (node->right->bit > node->bit)
        PrintDefinitions(node->right, out);
}

void TTrie::PrintRelations(TNode *node, std::ofstream &out)
{
    if (node->left->bit > node->bit) {
        out << ' ' << safeKey(node) << "->" << safeKey(node->left) <<
"[label=\"l\"];\n";
        PrintRelations(node->left, out);
    }
    else {
        out << ' ' << safeKey(node) << "->" << safeKey(node->left) <<
"[label=\"l\"];\n";
    }
    if (node->right->bit > node->bit) {
        out << ' ' << safeKey(node) << "->" << safeKey(node->right) <<
"[label=\"r\"];\n";
        PrintRelations(node->right, out);
    }
    else {
        out << ' ' << safeKey(node) << "->" << safeKey(node->right) <<
"[label=\"r\"];\n";
    }
}
[leo@pc ver1_static_fork]$ cat PatFuncs/Node.hpp
#pragma once

const int MAXLEN = 256;
typedef unsigned long long TValue;
typedef char TKey;

struct TNode
{
    int id = -1;
    int bit;

    TKey *key;
    TValue value;

    TNode *left;
    TNode *right;

    void Initialize(int b, TKey *k, TValue v, TNode *l, TNode *r);
```

```
    TNode();

    TNode(int b, TKey *k, TValue v);

    TNode(int b, TKey *k, TValue v, TNode *l, TNode *r);

    ~TNode();
};

[leo@pc ver1_static_fork]$ cat PatFuncs/Node.cpp
#include "Node.hpp"
#include <cstring>

void TNode::Initialize(int b, TKey *k, TValue v, TNode *l, TNode *r)
{
    bit = b;
    if (k) {
        key = new char[strlen(k) + 1];
        strcpy(key, k);
    }
    else
        key = k;
    value = v;
    left = l;
    right = r;
}

TNode::TNode()
{
    Initialize(-1, 0, 0, this, this);
}

TNode::TNode(int b, TKey *k, TValue v)
{
    Initialize(b, k, v, this, this);
}

TNode::TNode(int b, TKey *k, TValue v, TNode *l, TNode *r)
{
    Initialize(b, k, v, l, r);
}

TNode::~TNode()
{
    delete[] key;
}


[leo@pc ver1_static_fork]$ cat main.cpp
#include <bits/stdc++.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <unistd.h>
```

```
#include "PatFuncs/Additional.hpp"
#include "PatFuncs/Trie.hpp"

int main()
{
    std::ofstream fout;
    std::ofstream dotout;
    std::ifstream fin;

    char input[MAXLEN];
    TValue value;

    TTrie *trie;
    try {
        trie = new TTrie();
    }
    catch (const std::bad_alloc &e) {
        std::cout << "ERROR: fail to allocate the requested storage space\n";
        exit(1);
    }

    TNode *node;

    while ((std::cin >> input)) {
        if (!std::strcmp(input, "+")) {
            std::cin >> input;
            Lowercase(input);
            std::cin >> value;

            std::cout << (trie->Insert(input, value) ? "OK" : "Exist");
            std::cout << '\n';
        }
        else if (!std::strcmp(input, "-")) {
            std::cin >> input;
            Lowercase(input);

            std::cout << (trie->Delete(input) ? "OK" : "NoSuchWord");
            std::cout << '\n';
        }
        else if (!std::strcmp(input, "!")) {
            std::cin >> input;
            if (!std::strcmp(input, "Save")) {
                std::cin >> input;
                fout.open(input, std::ios::out | std::ios::binary |
std::ios::trunc);
                if (!fout.is_open()) {
                    std::cout << "ERROR: can't create file\n";
                    continue;
                }

                trie->Save(fout);
                std::cout << "OK\n";

                fout.close();
            }
```

```
            else if (!std::strcmp(input, "Load")) {
                std::cin >> input;
                fin.open(input, std::ios::in | std::ios::binary);
                if (!fin.is_open()) {
                    std::cout << "ERROR: can't open file\n";
                    continue;
                }

                delete trie;
                trie = new TTrie();
                trie->Load(fin);

                std::cout << "OK\n";

                fin.close();
            }
        }
        else if (!std::strcmp(input, "print")) {
            std::string fname, vwr;
            std::cin >> fname >> vwr;

            int pid1 = fork(), childStatus1;
            if (pid1 < 0) {
                perror("fork1 fails\n");
                exit(1);
            }
            else if (pid1 == 0) {
                // creating, fillin .dot
                std::ofstream dot;
                dot.open("source.dot", std::ios::out | std::ios::trunc);
                dot << "digraph {\n";
                trie->PrintDefinitions(trie->root, dot);
                trie->PrintRelations(trie->root, dot);
                dot << "}\n";
                dot.flush(), dot.close();

                // for generating/opening png
                int pid2 = fork(), childStatus2;
                if (pid2 < 0) {
                    perror("fork2 fails\n");
                    exit(1);
                }
                else if (pid2 == 0) {
                    if (execlp("./drawer", "drawer", fname.c_str(), vwr.c_str(),
NULL) < 0) {
                        perror("execlp fails");
                        exit(1);
                    }
                }
                waitpid(pid2, &childStatus2, 0);
                if (!WIFEXITED(childStatus2))
                    perror("Something is wrong with 'worker' process\n");
            }
            waitpid(pid1, &childStatus1, 0);
            if (!WIFEXITED(childStatus1))
```

```
                perror("Something is wrong with 'dot creating' process\n");
            std::cout.flush();
        }
        else if (!strcmp(input, "exit")) {
            break;
        }
        else {
            Lowercase(input);
            node = trie->Find(input);
            if (!node)
                std::cout << "NoSuchWord";
            else
                std::cout << "OK: " << node->value;
            std::cout << '\n';
        }
    }

    delete trie;

    return 0;
}
[leo@pc ver1_static_fork]$ cat draw.hpp
#pragma once

#include <string>

extern "C" void generate(std::string filename);
extern "C" void view(std::string viewer, std::string filename);[leo@pc
ver1_static_fork]$ cat draw1.cpp
#include "draw.hpp"

#include <string>

extern "C" void generate(std::string filename)
{
    std::string cmd = "dot source.dot -Tpng -o ";
    cmd += filename + ".png";
    if (system(cmd.c_str()) == -1) {
        perror("Couldn't generate .dot file\n");
    }
}

extern "C" void view(std::string viewer, std::string filename)
{
    std::string cmd = viewer + " " + filename + ".png 2> /dev/null";
    if (system(cmd.c_str()) == -1) {
        perror("Couldn't open image\n");
    }
}
[leo@pc ver1_static_fork]$ cat drawer.cpp
#include "draw.hpp"
#include <dlfcn.h>
#include <bits/stdc++.h>

int main(int argc, char **argv)
```

```
{
    if (argc < 3) {
        std::cout << "Not enough arguments for drawer-program\n";
        return 0;
    }

    std::string fname = argv[1];
    std::string vwr = argv[2];

    void *library_handler = NULL;
    library_handler = dlopen("./libdraw.so", RTLD_LAZY);
    if (!library_handler) {
        fprintf(stderr, "dlopen() error: %s\n", dlerror());
        exit(1);
    }

    void (*generate)(std::string filename);
    void (*view)(std::string viewer, std::string filename);

    generate = (void (*)(std::string))dlsym(library_handler, "generate");
    std::cout << dlerror() << std::endl;

    view = (void (*)(std::string, std::string))dlsym(library_handler, "view");
    std::cout << dlerror() << std::endl;

    generate(fname);
    view(vwr, fname);

    dlclose(library_handler);
    return 0;
}
[leo@pc ver1_static_fork]$ cat Makefile
all: drawer main

drawer: libdraw.so drawer.cpp
        g++ -g3 drawer.cpp -ldl -o drawer

libdraw.so: draw1.o
        g++ -shared draw1.o -o libdraw.so

draw1.o: draw1.cpp
        g++ -fPIC -c draw1.cpp -o draw1.o

main: main.cpp
        g++ main.cpp -lpatricia -L./ -o main

clean:
        rm -rf *.o main draw
```

9. Заключение.

С большим интересом мною было проведено небольшое исследование на тему сжатых префиксных деревьев, ибо алгоритмы и структуры данных — моя пассия. К моему удивлению, эта структура данных является запатентованной разработкой, поэтому её редко где изучают и по ней практически нет информации в интернете. Пришлось читать оригинальную статью автора 1967 года и другие книги по алгоритмам, что было мне в радость.

Признаться, собой я чрезвычайно доволен, ведь описал, во-первых, крайне практичную структуру данных, а во-вторых — закрепил все свои знания, навыки, полученные по мере прохождения курса Операционных Систем в этом семестре. Подобная синергия двух любимых вещей (проектирование алгоритмов и структур данных с операционными системами) принесла мне небывалый восторг.

В дополнении ко всему этому я, надеюсь, упростил жизнь будущим студентам. Они смогут запустить мою программу у себя и побаловаться с конструированием этого дерева, что позволит им лучше усваивать материал на эту тему.