МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по курсовой работе

по дисциплине «Алгоритмы и структуры данных»

Тема: Поиск и применение эффективных структур данных

Студент гр. 3388 _____ Снигирёв А.А.

Преподаватель _____ Шалагинов И.В.

Санкт-Петербург

2024

Цель работы

Изучить поставленную задачу и реализовать максимально подходящую структуру данных из ранее изученных для ее решения.

Задание

ВВариант 6

Вы работаете в компании, которая управляет большим количеством доменных имен и их соответствующими IP-адресами. Вам необходимо разработать систему, которая будет быстро и эффективно обрабатывать запросы на преобразование доменных имен в IP-адреса и наоборот.

Каждое доменное имя может быть связано с одним IP-адресом, и каждый IP-адрес может быть связан с одним доменным именем.

Ваша задача — создать программу, которая будет хранить и управлять этими данными. Программа должна поддерживать следующие операции:

Добавление доменного имени и IP-адреса в систему.

Удаление доменного имени и IP-адреса из системы.

Поиск IP-адреса по доменному имени.

Поиск доменного имени по IP-адресу.

Вывод всех доменных имен и их соответствующих IP-адресов.

Для выполнения данной задачи реализуйте наиболее подходящую структуру из изученных вами.

Исследование

Для решения задачи потенциально можно использовать любую структуру данных, но подходящими характеристиками обладают только балансированные деревья и хэш-таблицы.

Рассмотрим хэш-таблицы:

- Открытая адресация при больших объемах данных неудобна и медленна, поэтому будет рассматриваться только метод цепочек.
- В обычном методе цепочек в ячейках таблицы используется связный список. Операции в нем имеют сложность O(n), что может быть неоптимально, так как в задаче указано, что количество элементов очень велико.
- Поиск нужной ячейки происходит за константное время, что несомненно, является преимуществом

Рассмотрим АВЛ-деревья:

- Проще в реализации, чем КЧД, но балансировка в них происходит чаще, что не очень хорошо для частых вставок и удалений.
- Позволяют очень быстро проводить поиск элементов за время O(logn).
- Автоматически сортирует данные, что может быть полезно.

Как видно, обе структуры данных имеют свои преимущества и недостатки. Поэтому было решено объединить хэш-таблицы и АВЛ-деревья: в методе цепочек в каждой ячейке таблицы будет хранится АВЛ-дерево.

Также на выбор повлиял факт того, что в реальных системах по типу DNS намного чаще происходит преобразование ip в доменное имя, т.е поиск по ключу, а не вставка или удаление. Поэтому недостатки АВЛ-деревьев в виде затрат на балансировку будут не так критичны, а достоинство в виде быстрого поиска очень полезно.

Но, поскольку система должна работать в обе стороны, то недостатком является то, что придется для каждой ячейки использовать по два дерева: для

доменных имен и для адресов, что провоцирует использование дополнительной памяти. Однако скорость работы важнее, а еще появляется приятный бонус в виде улучшенной надежности системы.

Итоговая сложность операций:

- Вставка — O(1) + O(log(n)) = O(log(n))
- Удаление — O(1) + O(log(n)) = O(log(n))
- Поиск - O(log(n))

Выполнение работы

1. Класс HashTableIp

- Это модифицированная реализация хэш-таблицы, которая позволяет хранить данные в виде пар "ключ-значение".
- Методы:
  - add – добавляет пару ключ-значение в обе таблицы
  - delete_element - удаляет значение по ключу из обеих таблиц
  - search - извлекает элемент по ключу.
  - Print - Выводит содержимое хэш-таблицы для визуализации.

2. Класс Node

- Узел для АВЛ-дерева, содержит данные о доменном имене, ip-адресе, потомках, родителе и высоте.

3. Классы AVL, Tree_ip, Tree_domain – класс-родитель и два потомка, Tree_ip построено с использованием сравнения ip-адресов, а Tree_domain – доменных имен.

- Методы:
  AVL:
  - update_height: обновление высот узлов
  - left_rotate & right_rotate: левый и правый повороты для балансировки
  - in_order_traversal: Симметричный обход дерева для получения отсортированных данных.
  - print_tree: вывод элементов дерева
  - balance: балансировка.
  - get_balance: проверка сбалансированности
- Для восстановления баланса используются вращения

  Tree_ip и Tree_domain:

- insert_value: вставка

- delete_value: удаление
- search: поиск

4. Вспомогательные функции

- cmp_ip: возвращает результат сравнения двух строк ip адресов. Используется для построения бинарного дерева.

- cmp_domains: сравнивает доменные имена. Также используется для построения дерева.

- is_ip: проверяет, является ли строка ip-адресом. Используется для определения, по какому элементу пары ip-domain производить операцию.

- ip_hash и ip_to_int: используются для вычисления хэша ip-адреса

- fnv1a_hash – вычисляет хэш доменного имени.

Хэш-функции были подобраны исходя из количества коллизий и распределения элементов. Эти две показали себя лучше остальных.

Так, при заполнении таблицы размером 1000 10 тысячами случайных значений, дерево каждой ячейки хранило от 8 до 16 элементов, что является довольно хорошим распределением.

Исходный код см. в **ПРИЛОЖЕНИИ А.**

Тестирование. Для функций и методов были написаны два разных файла с тестами. Результаты тестирования см. в **ПРИЛОЖЕНИИ Б**

Также для использования структуры данных был разработан простой консольный интерфейс, позволяющий вводить запросы для управления структурой данных и получения значений.

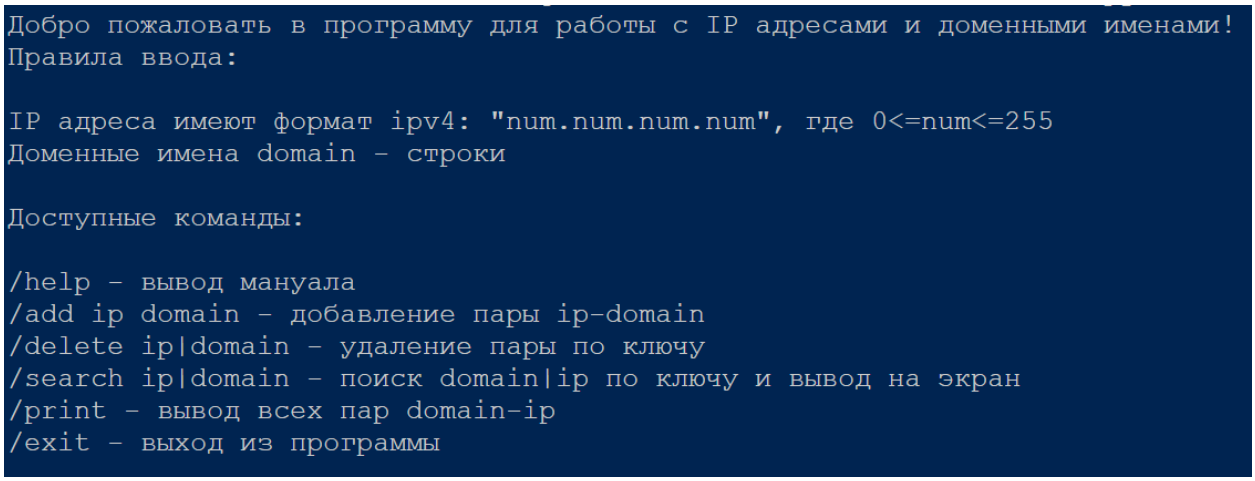

Рис. 1 — консольный интерфейс структуры данных.

Выводы

В ходе проделанной работы разработана система для работы с доменными именами и ip-адресами, использующая Хеш-таблицу+АВЛ-дерево для быстрого поиска, вставки и удаления элементов. Был реализован консольный интерфейс и написаны юнит-тесты.

ПРИЛОЖЕНИЕ А

main.py

```
from modules.ip_hash_table import HashTableIp
from modules.config import config
from cli import start


if __name__ == "__main__":
    start()
```

cli.py

```
from modules.ip_hash_table import HashTableIp, is_ip

def start():
    tbl = HashTableIp()
    with open("manual.txt") as file:
        help_msg = file.readlines()
    print(''.join(help_msg))

    while True:
        command = input()
        if command == "/exit":
            print("Exit..")
            break
        elif command == "/help":
            print(''.join(help_msg))
        elif command=='/print':
            tbl.print()
        else:
            operands = list(command.split())
            if operands[0]=='/add':
                if is_ip(operands[1])==False:
                    print("Ircorrect IP")
                else:
```

```
                tbl.add(operands[1], operands[2])
            elif operands[0]=='/delete':
                tbl.delete_pair(operands[1])
            elif operands[0]=='/search':
                print(tbl.search(operands[1]))
            else:
                print("Error input")
```

functions_tests.py и methods_tests.py

```
import pytest
from modules.ip_hash_table import HashTableIp

@pytest.fixture
def hash_table():
    return HashTableIp()

# Тест добавления и поиска IP
def test_add_and_search_ip(hash_table):
    hash_table.add("192.168.0.1", "example.com")
    hash_table.add("10.0.0.1", "test.org")

    assert hash_table.search("192.168.0.1") == "example.com"
    assert hash_table.search("10.0.0.1") == "test.org"
    assert hash_table.search("172.16.0.1") is None

# Тест добавления и поиска домена
def test_add_and_search_domain(hash_table):
    hash_table.add("192.168.0.1", "example.com")
    hash_table.add("10.0.0.1","test.org")

    assert hash_table.search("example.com") == "192.168.0.1"
    assert hash_table.search("test.org") == "10.0.0.1"
    assert hash_table.search("unknown.com") is None

# Тест удаления IP
```

```
def test_delete_ip(hash_table):
    hash_table.add("192.168.0.1", "example.com")
    hash_table.delete_ip("192.168.0.1")

    assert hash_table.search("192.168.0.1") is None

def test_delete_domain(hash_table):
    hash_table.add("192.168.0.1","example.com")
    hash_table.delete_domain("example.com")

    assert hash_table.search("example.com") is None

# Тест удаления пары
def test_delete_pair(hash_table):
    hash_table.add("192.168.0.1", "example.com")
    hash_table.add("10.0.0.1", "test.org")

    hash_table.delete_pair("192.168.0.1")
    assert hash_table.search("192.168.0.1") is None
    assert hash_table.search("10.0.0.1") == "test.org"

    hash_table.delete_pair("test.org")
    assert hash_table.search("10.0.0.1") is None

# Тест печати таблицы
def test_print_table(hash_table, capsys):
    hash_table.add("192.168.0.1", "example.com")
    hash_table.add("10.0.0.1", "test.org")

    hash_table.print()
    captured = capsys.readouterr()

    assert "192.168.0.1" in captured.out
    assert "example.com" in captured.out
```

```
    assert "10.0.0.1" in captured.out
    assert "test.org" in captured.out

import pytest
from modules.ip_hash_table import is_ip, ip_to_int,
ip_hash,fnv1a_hash

# Тест функции is_ip
@pytest.mark.parametrize("ip,expected", [
    ("192.168.0.1", True),
    ("10.0.0.1", True),
    ("example.com", False),
    ("invalid_ip", False),
])
def test_is_ip(ip, expected):
    assert is_ip(ip) == expected

# Тест функции ip_to_int
def test_ip_to_int():
    assert ip_to_int("192.168.0.1") == 3232235521
    assert ip_to_int("10.0.0.1") == 167772161

# Тест функции ip_hash
def test_ip_hash():
    table_size = 1009
            assert ip_hash("192.168.0.1", table_size) ==
ip_to_int("192.168.0.1") % table_size
             assert ip_hash("10.0.0.1", table_size) ==
ip_to_int("10.0.0.1") % table_size

# Тест функции fnv1a_hash
def test_fnv1a_hash():
    table_size = 1009
    domain = "example.com"
```

```
    result = fnv1a_hash(domain, table_size)
    assert isinstance(result, int)
    assert 0 <= result < table_size
```

AVL.py

```
class Node:
      def __init__(self, key, value, left=None, right=None) ->
None:
        self.val: str = value
        self.key: list[int] = key
        self.left: Node | None = left
        self.right: Node | None = right
        self.height: int = 1

class AVL:
    def __init__(self) -> None:
        self.root: Node | None = None

    @staticmethod
    def _get_height(node: Node | None) -> int:
        return node.height if node else 0

    def update_height(self, node: Node | None) -> None:
                node.height  =  max(self._get_height(node.left),
self._get_height(node.right)) + 1

    def _get_balance(self, node: Node | None) -> int:
                        return   self._get_height(node.right)   -
self._get_height(node.left) if node else 0

    def right_rotate(self, node: Node) -> Node:
        left_child = node.left
        buffer = left_child.right
        left_child.right = node
```

```
        node.left = buffer
        self.update_height(node)
        self.update_height(left_child)
        return left_child

    def left_rotate(self, node: Node) -> Node:
        right_child = node.right
        buffer = right_child.left
        right_child.left = node
        node.right = buffer
        self.update_height(node)
        self.update_height(right_child)
        return right_child

    def balance(self, node: Node) -> Node:
        self.update_height(node)
        cur_balance = self._get_balance(node)
        if cur_balance < -1:
            if self._get_balance(node.left) > 0:
                node.left = self.left_rotate(node.left)
            node = self.right_rotate(node)
        elif cur_balance > 1:
            if self._get_balance(node.right) < 0:
                node.right = self.right_rotate(node.right)
            node = self.left_rotate(node)
        return node

    def diff(self, root: Node | None) -> int:
        if root is None:
            return float("inf")
        ans = []
        if root.left is not None:
            ans.append(abs(root.left.val - root.val))
        if root.right is not None:
```

```
            ans.append(abs(root.right.val - root.val))
                    return   min(ans   +   [self.diff(root.left),
self.diff(root.right)])

      def  in_order_traversal(self,  node:  Node  |  None,  result:
list[int] | None = None) -> list[int]:
        if result is None:
            result = []
        if node is not None:
            self.in_order_traversal(node.left, result)
            result.append([node.key, node.val])
            self.in_order_traversal(node.right, result)
        return result
```

Tree_ip.py

```
from .AVL import AVL, Node


def cmp_ip(ip1, ip2):
    for i in range(len(ip1)):
        if ip1[i]>ip2[i]:
            return 1
        elif ip1[i]<ip2[i]:
            return -1
    return 0

class Tree_ip(AVL):
     def  _insert(self,  val:  str,  key:  list[int],  node:  Node  |
None) -> Node:
        if node is None:
            return Node(key, val)
        if cmp_ip(key, node.key)==-1:
            node.left = self._insert(val, key, node.left)
        elif cmp_ip(key, node.key)==1:
            node.right = self._insert(val, key, node.right)
```

```
        return self.balance(node)

    def insert_value(self, val: str, key: list[int]) -> None:
        self.root = self._insert(val, key, self.root)

    def delete_value(self, key: list[int]) -> None:
        self.root = self._delete(self.root, key)

    def _delete(self, node: Node | None, key: list[int]) -> Node
| None:
        if node is None:
            return node
        if cmp_ip(key, node.key)==-1:
            node.left = self._delete(node.left, key)
        elif cmp_ip(key, node.key)==1:
            node.right = self._delete(node.right, key)
        elif node.left is not None and node.right is not None:
            right_min = node.right
            while right_min.left is not None:
                right_min = right_min.left
            node.key = right_min.key
            node.val = right_min.val
            node.right = self._delete(node.right, node.key)
        elif node.left is not None:
            node = node.left
        elif node.right is not None:
            node = node.right
        else:
            node = None
        return self.balance(node) if node else None

    def search(self, key: list[int]) -> str | None:
        current = self.root
        while current is not None:
```

```
            comparison = cmp_ip(key, current.key)
            if comparison == 0:
                return current.val  # Найден нужный узел
            elif comparison == -1:
                current = current.left  # Идем влево
            else:
                current = current.right  # Идем вправо
        return None

    def print_tree(self) -> None:
        curr_list = self.in_order_traversal(self.root)
        for elem in curr_list:
                 print(".".join(list(map(str,elem[0]))), " - ",
elem[1])
```

Tree_domain.py

```
from .AVL import AVL, Node

def cmp_domains(str1: str, str2: str) -> int:
    if str1 < str2:
        return -1
    elif str1 > str2:
        return 1
    else:
        return 0

class Tree_domain(AVL):
     def _insert(self, val: str, key: str, node: Node | None) ->
Node:
        if node is None:
```

```
            return Node(key, val)
        if cmp_domains(key, node.key)==-1:
            node.left = self._insert(val, key, node.left)
        elif cmp_domains(key, node.key)==1:
            node.right = self._insert(val, key, node.right)
        return self.balance(node)

    def insert_value(self, val: str, key: str) -> None:
        self.root = self._insert(val, key, self.root)

    def delete_value(self, key: str) -> None:
        self.root = self._delete(self.root, key)

     def _delete(self, node: Node | None, key: str) -> Node |
None:
        if node is None:
            return node
        if cmp_domains(key, node.key)==-1:
            node.left = self._delete(node.left, key)
        elif cmp_domains(key, node.key)==1:
            node.right = self._delete(node.right, key)
        elif node.left is not None and node.right is not None:
            right_min = node.right
            while right_min.left is not None:
                right_min = right_min.left
            node.key = right_min.key
            node.val = right_min.val
            node.right = self._delete(node.right, node.key)
        elif node.left is not None:
            node = node.left
        elif node.right is not None:
            node = node.right
        else:
            node = None
```

```
        return self.balance(node) if node else None

    def search(self, key: str) -> str | None:
        current = self.root
        while current is not None:
            comparison = cmp_domains(key, current.key)
            if comparison == 0:
                return current.val  # Найден нужный узел
            elif comparison == -1:
                current = current.left  # Идем влево
            else:
                current = current.right  # Идем вправо
        return None
    def print_tree(self) -> None:
        curr_list = self.in_order_traversal(self.root)
        for elem in curr_list:
            print(elem[0], " - ", elem[1])
```

HashTableIp.py

```
from .config import config
from .ip_tree import Tree_ip
from .domain_tree import Tree_domain
import random
def ip_to_int(ip: str) -> int:
    parts = map(int, ip.split('.'))
    result = 0
    for part in parts:
        result = (result << 8) + part
    return result

def ip_hash(ip: str, table_size: int) -> int:
    ip_int = ip_to_int(ip)
    return ip_int % table_size

def fnv1a_hash(domain: str, table_size: int) -> int:
```

```
    fnv_prime = 16777619
     hash_value = 2166136261   # Начальное значение (FNV offset
basis)

    for char in domain:
        hash_value ^= ord(char)  # XOR текущего символа
          hash_value *= fnv_prime  # Умножение на фиксированное
простое число
            hash_value &= 0xffffffff  # Ограничение до 32 бит
(аналог переполнения)

    return hash_value % table_size


def is_ip(string: str):
    l = len(string)
    result = (string.count('.')==3 and l<16 and l>6)
    return result



class HashTableIp:
    def __init__(self):
                    self.table_ip  =  [Tree_ip()  for  _  in
range(config.table_size)]
                self.table_domain  =  [Tree_domain()  for  _  in
range(config.table_size)]
        self.max_len = config.table_size
    def add(self, key:str, value:str):
        ip_hsh = ip_hash(key,config.table_size)
        domain_hash = fnv1a_hash(value, config.table_size)
        key_lst = list(map(int, key.split('.')))
         self.table_ip[ip_hsh].insert_value(val=value,key=key_lst
)
```

```
            self.table_domain[domain_hash].insert_value(key=value,
val=key)

    def delete_ip(self, key:str, flag=True):
        key_hash = ip_hash(key,config.table_size)
        key_lst = list(map(int, key.split('.')))
        domain = self.table_ip[key_hash].search(key_lst)
        self.table_ip[key_hash].delete_value(key_lst)
        if flag==True:
            self.delete_domain(domain)

    def delete_domain(self, domain, flag = True):
        domain_hash = fnv1a_hash(domain, config.table_size)
        ip = self.table_domain[domain_hash].search(domain)
        self.table_domain[domain_hash].delete_value(domain)
        if flag == True:
            self.delete_ip(ip, False)

    def delete_pair(self, key:str):
        try:
            if is_ip(key):
                self.delete_ip(key)
            else:
                self.delete_domain(key)
        except AttributeError:
            print("Error input")

    def search(self, key: str):
        if is_ip(key):
            key_hash = ip_hash(key,config.table_size)
            key = list(map(int, key.split('.')))
            return self.table_ip[key_hash].search(key)
        key_hash = fnv1a_hash(key,config.table_size)
        return self.table_domain[key_hash].search(key)
```

```
def print(self):
    print("Domain Name      IP")
    for tree in self.table_domain:
        tree.print_tree()
    print('\n')
```

ПРИЛОЖЕНИЕ Б

Тестирование

```
root@M14-I3W302:/mnt/c/Users/snigi/OneDrive/Рабочий стол/CW3/src# pytest methods_tests.py
==================================== test session starts ====================================
platform linux -- Python 3.10.12, pytest-8.3.3, pluggy-1.5.0
rootdir: /mnt/c/Users/snigi/OneDrive/Рабочий стол/CW3/src
collected 6 items

methods_tests.py ......                                                              [100%]

==================================== 6 passed in 0.13s ======================================
root@M14-I3W302:/mnt/c/Users/snigi/OneDrive/Рабочий стол/CW3/src#
```

Рис. 2 — Тестирование методов классов

```
root@M14-I3W302:/mnt/c/Users/snigi/OneDrive/Рабочий стол/CW3/src# pytest functions_tests.py
==================================== test session starts ====================================
platform linux -- Python 3.10.12, pytest-8.3.3, pluggy-1.5.0
rootdir: /mnt/c/Users/snigi/OneDrive/Рабочий стол/CW3/src
collected 7 items

functions_tests.py .......                                                           [100%]

==================================== 7 passed in 0.25s ======================================
```

Рис. 3 — Тестирование вспомогательных функций

Литература:

https://habr.com/ru/articles/509220/ - статья про хэш-таблицы

https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%92%D0%9B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE — статья про авл-деревья

https://www.python.org/ - использовался для поиска некоторых функций