ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

доцент		С. А. Чернышев
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 7

ТЕЛЕФОННАЯ КНИГА

по курсу:

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 4326		Г. С. Томчук
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

СОДЕРЖАНИЕ

1 Цель работы

Целью данной лабораторной работы является создать структуру данных для телефонной книги, поддерживающую операции добавления, удаления и поиска контактов по номеру. Реализовать основные операции с учетом ограничений на номера и имена, а также два дополнительных задания: поиск по частичному совпадению номера и поддержка групп и меток для контактов.

2 Задание

Разработать программу, которая:

1. Обрабатывает команды:
   - add number name — добавляет контакт. Если номер уже существует, обновляет имя.
   - del number — удаляет контакт по номеру.
   - find number — находит имя по номеру или выводит «not found».
2. Реализует два дополнительных задания:
   - Поиск по частичному совпадению номера.
   - Поддержку групп и меток для контактов.

Программа должна быть оптимизирована для работы с большими данными.

3 Краткое описание хода разработки

1. Изучены ограничения задачи, такие как длина номера, допустимые символы в имени, и количество запросов. Определено, что для хранения данных подойдет словарь (хеш-таблица), поскольку он обеспечивает доступ к данным за O(1) в среднем случае.
2. Для основной задачи создан класс PhoneBook, содержащий словарь для хранения пар "номер-имя". Дополнительно реализованы структуры для хранения меток и групп.
3. Основной функционал (добавление, удаление, поиск по номеру)

реализован через методы класса:

- add_contact: Добавляет новый контакт или обновляет существующий.
- delete_contact: Удаляет контакт, если он существует.
- find_contact: Возвращает имя или «not found», если номер отсутствует.

4. Реализованы два дополнительных задания:

- Поиск по частичному совпадению номера: Используется метод find_partial, который находит все номера, начинающиеся с заданной строки.
- Поддержка групп и меток: Контакты можно добавлять в группы и назначать им метки. Реализованы методы для поиска по группе или метке.

5. Запросы собираются в список, затем обрабатываются пакетно. Это позволяет поддерживать требуемый формат вывода, где результаты поиска выводятся после обработки всех запросов.
6. Программа протестирована на данных с разным количеством запросов, чтобы убедиться в корректной работе и соблюдении временных ограничений.

4 Исходный код программы

```
class PhoneBook:
    def __init__(self):
        self.contacts = {}  # Основной справочник {номер: {name, group, tags}}
        self.groups = {}  # Словарь групп {group_name: [numbers]}
        self.tags = {}  # Словарь меток {tag_name: [numbers]}

    def add_contact(self, number, name, group=None):
        if number in self.contacts:
            self.contacts[number]['name'] = name
        else:
            self.contacts[number] = {'name': name, 'group': group, 'tags':
set()}
        if group:
            if group not in self.groups:
                self.groups[group] = []
            if number not in self.groups[group]:
                self.groups[group].append(number)

    def delete_contact(self, number):
        if number in self.contacts:
```

```
            group = self.contacts[number]['group']
            if group and number in self.groups.get(group, []):
                self.groups[group].remove(number)
            for tag in self.contacts[number]['tags']:
                if number in self.tags[tag]:
                    self.tags[tag].remove(number)
            del self.contacts[number]

    def find_contact(self, number):
        return self.contacts.get(number, {}).get('name', 'not found')

    def add_tag(self, number, tag_name):
        if number in self.contacts:
            self.contacts[number]['tags'].add(tag_name)
            if tag_name not in self.tags:
                self.tags[tag_name] = []
            if number not in self.tags[tag_name]:
                self.tags[tag_name].append(number)

    def find_by_group(self, group_name):
        if group_name in self.groups:
            return [self.contacts[number]['name'] for number in
self.groups[group_name]]
        return []

    def find_by_tag(self, tag_name):
        if tag_name in self.tags:
            return [self.contacts[number]['name'] for number in
self.tags[tag_name]]
        return []

    def find_partial(self, partial_number):
        result = []
        for number in self.contacts:
            if str(number).startswith(partial_number):
                result.append((number, self.contacts[number]['name']))
        return result


def process_requests(requests):
    phone_book = PhoneBook()
    results = []
    for request in requests:
        query = request.split()
        if query[0] == "add":
            number, name = query[1], query[2]
            group = query[3] if len(query) > 3 else None
            phone_book.add_contact(number, name, group)
        elif query[0] == "del":
            number = query[1]
            phone_book.delete_contact(number)
        elif query[0] == "find":
            if len(query) == 2:
                number = query[1]
                results.append(phone_book.find_contact(number))
            elif query[1] == "partial":
                partial_number = query[2]
                matches = phone_book.find_partial(partial_number)
                if matches:
                    results.extend([f"{number}: {name}" for number, name in
matches])
                else:
```

```
                    results.append("not found")
            elif query[0] == "tag":
                number, tag_name = query[1], query[2]
                phone_book.add_tag(number, tag_name)
            elif query[0] == "find_by_group":
                group_name = query[1]
                names = phone_book.find_by_group(group_name)
                results.append(", ".join(names) if names else "not found")
            elif query[0] == "find_by_tag":
                tag_name = query[1]
                names = phone_book.find_by_tag(tag_name)
                results.append(", ".join(names) if names else "not found")
        return results


def main():
    q = int(input())
    requests = []
    for _ in range(q):
        requests.append(input())
    results = process_requests(requests)
    print("\n".join(results))


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

5 Реализация основного и двух любых дополнительных заданий из списка

Дополнительное задание 1: Поиск по частичному совпадению номера. Метод find_partial итерируется по ключам словаря, чтобы найти номера, начинающиеся с заданной подстроки.

Дополнительное задание 2: Поддержка групп и меток. Для реализации групп и меток добавлены дополнительные словари. Методы add_to_group и add_tag назначают контактам группы и метки, а find_by_group и find_by_tag позволяют искать контакты по этим критериям.

6 Результаты работы программы с примерами

На рис. 1, 2, 3 изображено тестирование написанной программы с различными входными данными.

```
8
find 2139803425
add 123456 me
add 0 granny
find 0
find 123456
del 0
del 0
find 0
not found
granny
me
not found

Process finished with exit code 0
```

Рисунок 1

```
12
add 911 police
add 76213 Mom
add 17239 Bob
find 76213
find 910
find 911
del 910
del 911
find 911
find 76213
add 76213 daddy
find 76213
Mom
not found
police
not found
Mom
daddy

Process finished with exit code 0
```

Рисунок 2

```
6
add 123 mom family
add 999 boss
add 456 dad family
tag 999 work
find_by_group family
find_by_tag work
mom, dad
boss

Process finished with exit code 0
```

Рисунок 3

7 Выводы

- Создана эффективная структура данных для управления телефонной книгой, соответствующая всем базовым требованиям задачи.
- Поиск по частичному совпадению номера облегчает работу с большим количеством контактов.
- Группы и метки позволяют структурировать телефонную книгу, добавляя гибкость в ее использование.
- Использование словарей обеспечивает высокую производительность операций добавления, удаления и поиска.
- Навыки, полученные в процессе работы:
    - Проектирование структур данных.
    - Реализация алгоритмов для сложных запросов.
    - Оптимизация работы программы с учетом ограничений.