Лабораторная работа 6 - Хеширование

Цель

Разобрать идею хеш-таблицы и применить хеширование в задачах на множество и словарь, а также в задаче на подсчет значений по ключу.

Выбор задания по хеш-функции

В данной лабораторной работе одна из задач выбирается по формуле:

```
H(v) = (A * v mod p) mod 9
```

где:

- A - последние две цифры номера группы,
- v - номер варианта в списке,
- p - следующее простое число, большее количества человек в группе.

Мой вариант v = 16.
Последние две цифры номера группы: 40, значит A = 40. (K3240)

Так как у меня академическая разница и я перевёлся с другого направления, фактического списка группы нет, поэтому количество человек было выбрано произвольно: 20.

Следующее простое число больше 20 равно 23, значит p = 23.

Считаем:

A * v = 40 * 16 = 640
640 mod 23 = 19
19 mod 9 = 1

Получаем H(v) = 1.

Нумерация задач ведётся с 1, поэтому номер задачи: t = H(v) + 1 = 2.

Таким образом, по хеш-функции выбрана задача №2. Дополнительно были выбраны задачи №1 и №5.

Задания и решения

Задание 1 - Множество

Нужно поддержать операции:

- `A x` - добавить `x`
- `D x` - удалить `x`
- `? x` - проверить наличие `x`

Сделал свое множество на хеш-таблице:

- открытая адресация (open addressing)
- линейное пробирование (linear probing)
- ленивое удаление через массив состояний: `0` - пусто, `1` - занято, `2` - удалено
- хеш-функция по лекции: `h(x) = ((a*x + b) mod p) mod m`, где `p` - большое простое, `m` - размер таблицы

```
class HashSetOA:
    _EMPTY = None

    def __init__(self, cap):
        self.cap = cap
        self.keys = [self._EMPTY] * cap
        self.states = bytearray(cap) # 0 empty, 1 filled, 2 deleted
        self.size = 0

        # универсальная хеш-функция из лекции: h(x) = ((a*x + b) mod p) mod
m
        # p берем большим простым > всех ключей по модулю, чтобы работало
стабильно
        self.p = (1 << 61) - 1
        self.a = 1000003
        self.b = 1000033

    def _h(self, x):
        # приводим к [0..p-1]
        x %= self.p
        return ((self.a * x + self.b) % self.p) % self.cap

    def _find_slot(self, x):
        j = self._h(x)
        first_del = -1

        for _ in range(self.cap):
            st = self.states[j]
            if st == 0:
                return (first_del if first_del != -1 else j, False)
            if st == 1 and self.keys[j] == x:
                return (j, True)
            if st == 2 and first_del == -1:
                first_del = j
            j += 1
            if j == self.cap:
                j = 0
        return (-1, False)

    def add(self, x):
        idx, exists = self._find_slot(x)
        if exists:
            return
        self.keys[idx] = x
        self.states[idx] = 1
        self.size += 1
```

```
    def discard(self, x):
        idx, exists = self._find_slot(x)
        if not exists:
            return
        self.states[idx] = 2
        self.size -= 1

    def contains(self, x):
        _, exists = self._find_slot(x)
        return exists


def main():
    with open("n1/input.txt", "rb") as f:
        data = f.read().split()

    n = int(data[0])
    i = 1

    cap = 1
    need = n * 2 + 1
    while cap < need:
        cap <<= 1

    hs = HashSetOA(cap)
    out = []

    for _ in range(n):
        op = data[i].decode()
        x = int(data[i + 1])
        i += 2

        if op == "A":
            hs.add(x)
        elif op == "D":
            hs.discard(x)
        else: # "?"
            out.append("Y" if hs.contains(x) else "N")

    with open("n1/output.txt", "w", encoding="utf-8") as f:
        f.write("\n".join(out))


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

Проверено на нескольких input-файлах (включая случаи с повторными вставками, удалениями и большим количеством запросов) - работает корректно.

Задание 2 - Телефонная книга

Команды:

- `add number name`
- `del number`
- `find number`

Сделал ассоциативный массив `number -> name` на хеш-таблице:

- открытая адресация + линейное пробирование
- приоритет на простоту: фиксированные `a` и `b`, `p` - большое простое
- удаление - через отметку `deleted`, чтобы не ломать поиск по пробированию

```
class HashMapOA:
    _EMPTY = None

    def __init__(self, cap):
        self.cap = cap
        self.keys = [self._EMPTY] * cap
        self.vals = [self._EMPTY] * cap
        self.states = bytearray(cap)  # 0 empty, 1 filled, 2 deleted
        self.size = 0

        # универсальная хеш-функция из лекции: h(x) = ((a*x + b) mod p) mod
m
        self.p = (1 << 61) - 1
        self.a = 1000003
        self.b = 1000033

    def _h(self, x):
        x %= self.p
        return ((self.a * x + self.b) % self.p) % self.cap

    def _find_slot(self, k):
        j = self._h(k)
        first_del = -1

        for _ in range(self.cap):
            st = self.states[j]
            if st == 0:
                return (first_del if first_del != -1 else j, False)
            if st == 1 and self.keys[j] == k:
                return (j, True)
            if st == 2 and first_del == -1:
                first_del = j
            j += 1
            if j == self.cap:
                j = 0
        return (-1, False)

    def put(self, k, v):
        idx, exists = self._find_slot(k)
        if not exists:
```

```
                self.size += 1
                self.keys[idx] = k
                self.states[idx] = 1
            self.vals[idx] = v

        def remove(self, k):
            idx, exists = self._find_slot(k)
            if not exists:
                return
            self.states[idx] = 2
            self.size -= 1

        def get(self, k):
            idx, exists = self._find_slot(k)
            if not exists:
                return None
            return self.vals[idx]


    def main():
        with open("n2/input.txt", "rb") as f:
            data = f.read().split()

        n = int(data[0])
        i = 1

        cap = 1
        need = n * 2 + 1
        while cap < need:
            cap <<= 1

        mp = HashMapOA(cap)
        out = []

        for _ in range(n):
            cmd = data[i].decode()

            if cmd == "add":
                number = int(data[i + 1])
                name = data[i + 2].decode()
                mp.put(number, name)
                i += 3
            elif cmd == "del":
                number = int(data[i + 1])
                mp.remove(number)
                i += 2
            else: # find
                number = int(data[i + 1])
                name = mp.get(number)
                out.append(name if name is not None else "not found")
                i += 2

        with open("n2/output.txt", "w", encoding="utf-8") as f:
            f.write("\n".join(out))
```

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

Проверено на нескольких input-файлах (добавление поверх старого ключа, удаление несуществующего, поиск после удалений) - вывод совпадает с ожидаемым.

Задание 5 - Выборы в США

На входе пары `name count`. Нужно суммировать голоса по каждому имени и вывести результат в лексикографическом порядке.

Тут тема хеша реализуется через стандартный `dict`:

- `dict` - это хеш-таблица
- операции `get/put` в среднем работают за `O(1)`, поэтому суммирование идет быстро

```
def main():
    with open("n3/input.txt", "rb") as f:
        data = f.read().split()

    votes = {}
    i = 0
    n = len(data)

    while i + 1 < n:
        name = data[i].decode()
        cnt = int(data[i + 1])
        votes[name] = votes.get(name, 0) + cnt
        i += 2

    keys = sorted(votes.keys())
    out = [f"{k} {votes[k]}" for k in keys]

    with open("n3/output.txt", "w", encoding="utf-8") as f:
        f.write("\n".join(out))


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

Проверено на нескольких input-файлах (разные имена, много повторов одного имени, пустые строки в конце файла) - работает корректно.