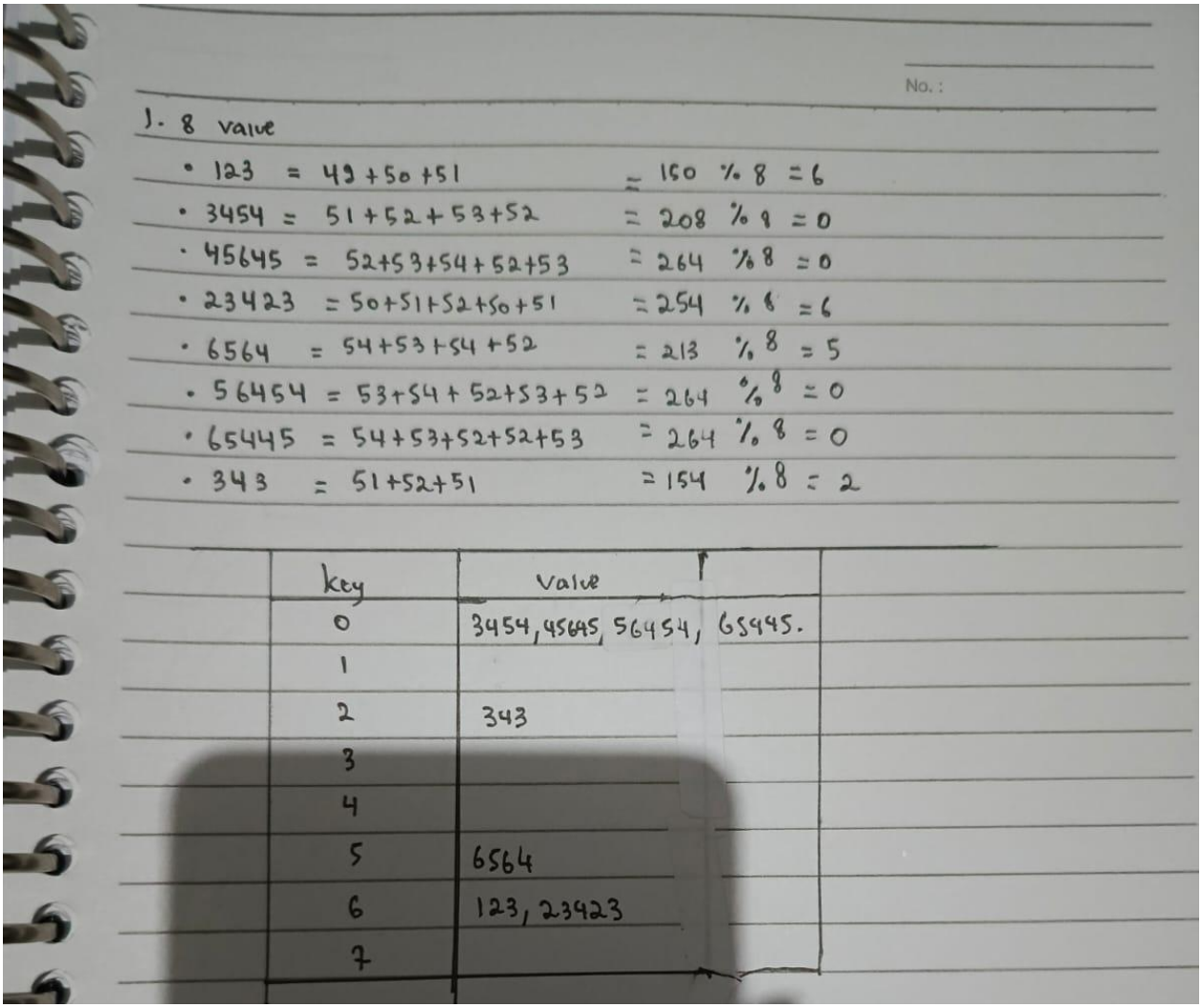
	LABORATORIUM SISTEM INFORMASI UNIVERSITAS TANJUNGPURA PONTIANAK Gedung FMIPA Jl. Prof. Dr. Hadari Nawawi Pontianak		
	Hari/Tanggal: Kamis/24 April 2025		
Nama Mahasiswa	Rafli Pratama	**Mata Kuliah Praktikum**	Algoritma dan struktur data
NIM	H1101241008	**Dosen Pengampu**	Ilhamsyah S.Si., M.Cs
Semester	2 Genap	**Paraf Dosen Pengampu**	
Kelas	Sisfo A	**Asisten Praktikum**	
Nilai		**Paraf Asisten Praktikum**	
LEMBAR KERJA PRAKTIKUM			

MATERI PRAKTIKUM : Hash Table

Percobaan 1.




```python
class HashTable:
    def __init__(self, size):
        self.size = size
        self.table = {}
        for i in range(size):
            self.table[i] = []

    def hash (self, key):
        return key % self.size

    def get(self,key):
        return self.table[self.hash(key)]

    def put (self, key, value):
        bucket = self.table[self.hash(key)]
        if value not in bucket:
            bucket.append(value)
```

```
table1 = HashTable(8)

daftarNama=[
'123',
'3454',
'45645',
'23423',
'6564',
'56454',
'65445',
'343'
]

for nama in daftarNama:
    key=sum(map(ord, nama))
    table1.put(key, nama)

for key in table1.table.keys():
    print(key, table1.table[key])
```

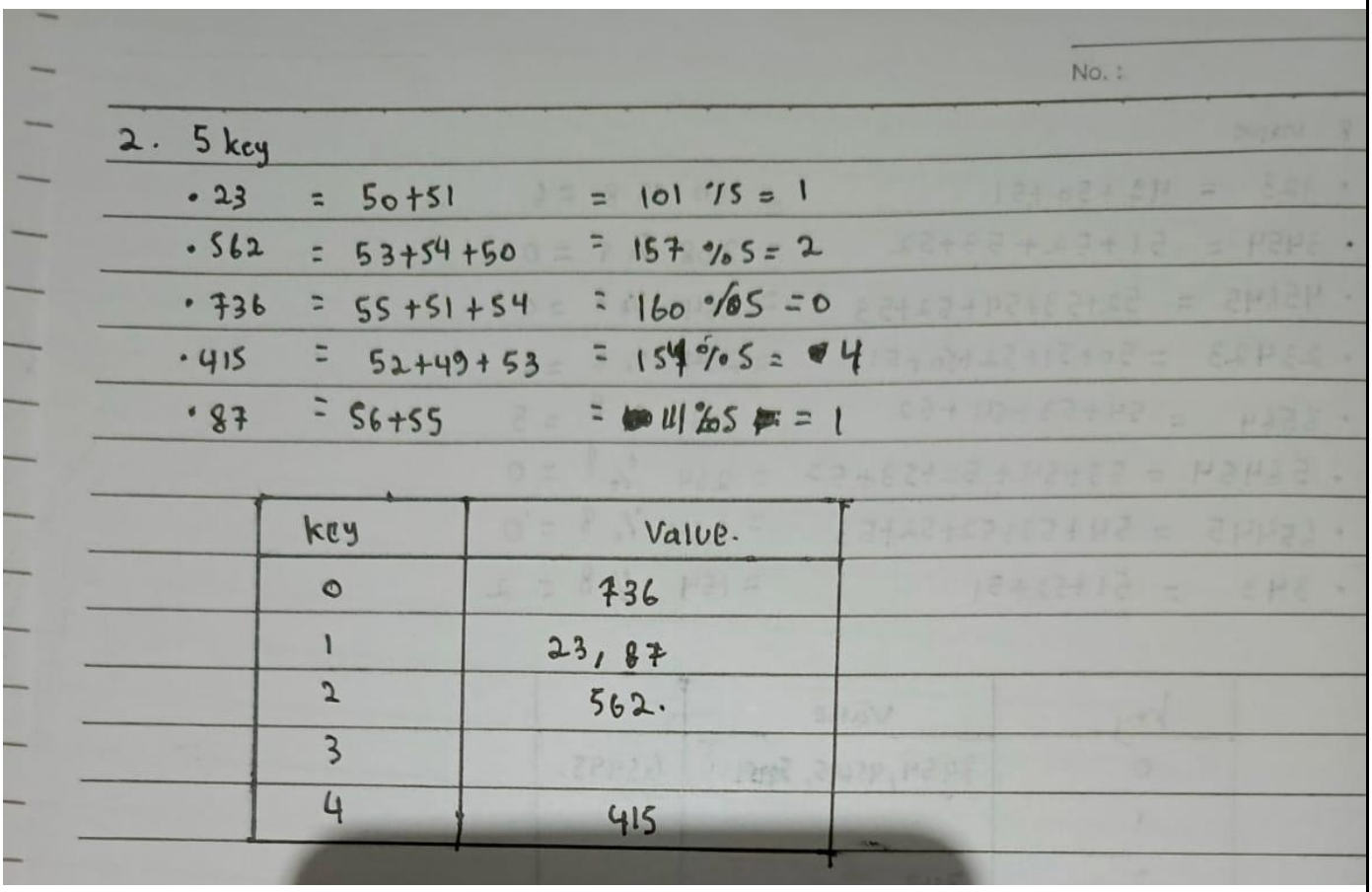
Output Program :
0 ['3454', '45645', '56454', '65445']
1 []
2 ['343']
3 []
4 []
5 ['6564']
6 ['123', '23423']
7 []

Penjelasan :

Sesuai dengan hasil yang di hitung secara manual
Pada program ini, dibuat sebuah kelas bernama HashTable yang bertujuan untuk mengimplementasikan struktur data hash table sederhana dengan metode chaining untuk menangani kemungkinan terjadinya collision. Kelas ini memiliki konstruktor __init__ yang menerima parameter size sebagai ukuran dari hash table. Dalam konstruktor tersebut, dilakukan inisialisasi dictionary self.table, di mana setiap indeks dari 0 hingga size-1 diisi dengan list kosong sebagai bucket penyimpanan data.
Fungsi hash(self, key) digunakan untuk menghasilkan indeks hash dengan cara menghitung sisa bagi dari key terhadap ukuran tabel (key % self.size). Fungsi ini bertugas menentukan lokasi penyimpanan nilai pada tabel. Selanjutnya, fungsi get(self, key) digunakan untuk mengambil data dari bucket sesuai indeks yang dihasilkan oleh fungsi hash. Fungsi put(self, key, value) digunakan untuk menyisipkan data ke dalam hash table. Jika nilai yang ingin dimasukkan belum ada dalam bucket terkait, maka nilai tersebut akan ditambahkan.

Selanjutnya, dibuat objek table1 dari kelas HashTable dengan ukuran tabel sebanyak 8 bucket. Data yang akan dimasukkan berupa daftar string yang disimpan dalam list daftarNama. Untuk setiap string dalam daftar tersebut, nilai kunci dihitung berdasarkan jumlah nilai ASCII dari masing-masing karakter dalam string menggunakan sum(map(ord, nama)). Nilai kunci tersebut kemudian digunakan dalam fungsi put() untuk menyimpan data ke dalam tabel.
Terakhir, program mencetak seluruh isi dari hash table dengan menampilkan indeks beserta isi bucket-nya. Dari hasil percobaan ini, dapat dipelajari bagaimana prinsip kerja hash table dalam menyimpan data, bagaimana proses hashing dilakukan, serta bagaimana menangani collision dengan metode chaining menggunakan list. Program ini juga menunjukkan pentingnya pemilihan fungsi hash yang baik agar distribusi data dalam tabel menjadi merata.

Percobaan 2.




```
class HashTable:
    def __init__(self, size):
        self.size = size
        self.table = {i: None for i in range(size)}

    def hash(self, key):
        return key % self.size

    def put(self, key, value):
```

```
        index = self.hash(key)
        original_index = index

        while self.table[index] is not None:
            if self.table[index] == value:
                return
            index = (index + 1) % self.size
            if index == original_index:
                raise Exception("Hash Table penuh!")

        self.table[index] = value

    def get(self, key):
        index = self.hash(key)
        original_index = index

        while self.table[index] is not None:
            if sum(map(ord, self.table[index])) == key:
                return self.table[index]
            index = (index + 1) % self.size
            if index == original_index:
                break
        return None

    def display(self):
        for key in self.table:
            print(f"{key}: {self.table[key]}")


table1 = HashTable(5)
# inisialisasi instance tabel dengan size 5

daftarNama=[
'23',
'562',
'736',
'415',
'87',
]

for nama in daftarNama:
    key=sum(map(ord, nama))
    table1.put(key, nama)

for key in table1.table.keys():
    print(key,[ table1.table[key]])
```

Output Program :
0 ['736']
1 ['23']
2 ['562']
3 ['87']
4 ['415']

Penjelasan :
Kalau mengikuti hasil hitung manual key yang mengalami collusion adalah key dengan index 1, maka dari itu dalam program kita akan menyelesaikan masalahnya.

Pada program ini, diimplementasikan struktur data Hash Table menggunakan metode linear probing sebagai teknik penanganan collision (tabrakan data). Kelas yang dibuat diberi nama HashTable, dan memiliki konstruktor __init__ yang menerima parameter size sebagai ukuran hash table. Setiap indeks dalam tabel diinisialisasi dengan nilai None sebagai penanda bahwa bucket masih kosong.
Fungsi hash(self, key) digunakan untuk menentukan indeks hash dari suatu *key*, dengan cara mengambil sisa pembagian dari key terhadap self.size. Fungsi ini menjadi dasar dalam menentukan lokasi penyimpanan nilai dalam tabel.
Fungsi put(self, key, value) bertugas untuk menyisipkan data ke dalam hash table. Jika posisi hasil hash sudah terisi, maka akan dilakukan pencarian slot kosong berikutnya secara berurutan (linear probing) hingga ditemukan slot kosong. Apabila ditemukan bahwa slot tersebut telah berisi data yang sama, maka proses penyisipan dihentikan untuk menghindari duplikasi. Bila seluruh tabel sudah terisi dan tidak ditemukan slot kosong, maka program akan mengeluarkan exception bahwa hash table penuh.
Fungsi get(self, key) digunakan untuk mencari data berdasarkan key yang telah dihitung sebelumnya. Proses pencarian dilakukan dengan memeriksa setiap slot sesuai urutan linear probing hingga data ditemukan atau hingga kembali ke posisi awal.
Fungsi display(self) dibuat untuk menampilkan isi hash table dengan mencetak setiap indeks beserta data yang tersimpan di dalamnya.
Dalam percobaan ini, dibuat instance table1 dari kelas HashTable dengan ukuran tabel sebanyak 5. Data yang disisipkan merupakan daftar string pada list daftarNama. Untuk setiap string dalam daftar tersebut, dilakukan perhitungan nilai key menggunakan jumlah nilai ASCII dari setiap karakter, yaitu sum(map(ord, nama)). Key ini kemudian digunakan untuk memasukkan data ke dalam hash table menggunakan metode linear probing.