

LAPORAN PRAKTIKUM

STRUKTUR DATA

MODUL KE-09

HASHING DALAM PYTHON



Disusun Oleh:

Nama : Oktario Mufti Yudha

NPM : 2320506044

Kelas : 04 (Empat)

Program Studi S1 Teknologi Informasi

Fakultas Teknik, Universitas Tidar

Genap 2023/2024

• Tujuan Praktikum

Adapun tujuan praktikum ini sebagai berikut :

- a. Mahasiswa akan belajar tentang konsep dasar Hashing, pada bahasa pemrograman Python.
- b. Mahasiswa akan mempelajari cara Representasi dari Struktur Data Hash, Table Hash, Fungsi Hash, Hashlib.

• Dasar Teori

Hashing adalah prinsip dasar dalam struktur data yang dapat menyimpan dan mengambil data dengan efisien, memungkinkan akses yang cepat. Hashing dilakukan dengan cara memetakan data ke indeks tertentu dalam tabel hash menggunakan fungsi hash, yang memungkinkan pengambilan informasi secara cepat berdasarkan kunci yang diberikan. Teknik ini sering digunakan dalam berbagai aplikasi pemrograman, termasuk basis data dan sistem caching, untuk meningkatkan efisiensi dalam pencarian dan pengambilan data.

Hashing memiliki tiga komponen utama:

- Kunci: Sebuah nilai yang bisa berupa string atau bilangan bulat yang digunakan sebagai input dalam fungsi hash untuk menentukan indeks atau lokasi penyimpanan dalam struktur data.
- Fungsi Hash: Fungsi yang menerima kunci input dan mengembalikan indeks elemen dalam tabel hash, yang dikenal sebagai indeks hash.
- Tabel Hash: Struktur data yang memetakan kunci ke nilai menggunakan fungsi hash, dengan menyimpan data secara asosiatif dalam array di mana setiap nilai memiliki indeks uniknya sendiri.

- Hasil dan Pembahasan

Table Hash

```
# Table Hash
ukuran = 20
class DataItem:
    def __init__(self, key, data):
        self.key = key
        self.data = data

hashArray = [None] * ukuran
dummyItem = DataItem(-1, -1)
item = None

def hashCode(key):
    return key % ukuran

def search(key):
    hashIndex = hashCode(key)
    while hashArray[hashIndex] != None:
        if hashArray[hashIndex].key == key:
            return hashArray[hashIndex]
        hashIndex = (hashIndex + 1) % ukuran
    return None

def insert(key, data):
    item = DataItem(key, data)
    hashIndex = hashCode(key)
    while hashArray[hashIndex] != None and hashArray[hashIndex].key != -1:
        hashIndex = (hashIndex + 1) % ukuran
    hashArray[hashIndex] = item

def deleteItem(item):
    key = item.key
    hashIndex = hashCode(key)
    while hashArray[hashIndex] != None:
        if hashArray[hashIndex].key == key:
            temp = hashArray[hashIndex]
            hashArray[hashIndex] = dummyItem
            return temp
        hashIndex = (hashIndex + 1) % ukuran
    return None

# Function to display the hash table
def display():
    for i in range(ukuran):
        if hashArray[i] != None:
            print("{} {}".format(hashArray[i].key, hashArray[i].data), end=" ")
        else:
            print("--", end=" ")
    print()

if __name__ == "__main__":
    insert(1, 20)
    insert(2, 70)
    insert(42, 80)
    insert(4, 25)
    insert(12, 44)
    insert(14, 32)
    insert(17, 11)
    insert(13, 78)
    insert(37, 97)

    print("insertion done")
    print("hash table content: ")
    display()
    item = search(37)
    if item != None:
        print("Element found: ", item.data)
    else:
        print("Element not found")
    deleteItem(item)
    item = search(37)
    if item != None:
        print("Element found: ", item.data)
    else:
        print("Element not found")
```

✓ 0.0s

```
insertion done
hash table content:
-- (1, 20) (2, 70) (42, 80) (4, 25) -- -- -- -- -- -- (12, 44) (13, 78) (14, 32) --
Element found: 97
Element not found
```

(Gambar 3.1)

- ukuran = 20: Menetapkan ukuran tabel hash menjadi 20.
- class DataItem:: Mendefinisikan kelas DataItem yang akan digunakan untuk menyimpan item data dalam tabel hash.
- def __init__(self, key, data):: Ini adalah konstruktor kelas DataItem. Ini mengambil kunci dan data sebagai argumen.
- self.key = key dan self.data = data: Menyimpan kunci dan data ke dalam instance DataItem.
- hashArray = [None] * ukuran: Membuat array dengan ukuran yang ditentukan dan mengisi semua elemen dengan None. Array ini akan digunakan sebagai tabel hash.
- dummyItem = DataItem(-1, -1): Membuat item dummy yang akan digunakan untuk menandai slot yang telah dihapus dalam tabel hash.
- item = None: Membuat variabel item dan mengatur nilainya menjadi None. Variabel ini akan digunakan nanti untuk menyimpan item yang dicari atau dihapus.
- def hashCode(key):: Mendefinisikan fungsi hashCode yang mengambil kunci sebagai argumen dan mengembalikan kode hash.
- return key % ukuran: Mengembalikan sisa pembagian kunci dengan ukuran tabel hash. Ini adalah fungsi hash sederhana.
- def search(key):: Mendefinisikan fungsi search yang mencari item dengan kunci tertentu dalam tabel hash.
- hashIndex = hashCode(key): Menghitung indeks hash dari kunci menggunakan fungsi hashCode.
- while hashArray[hashIndex] != None:: Melakukan loop selama slot tabel hash pada indeks hash tidak None.
- if hashArray[hashIndex].key == key:: Jika kunci item pada slot tabel hash sama dengan kunci yang dicari, maka item tersebut ditemukan.
- return hashArray[hashIndex]: Mengembalikan item yang ditemukan.
- hashIndex = (hashIndex + 1) % ukuran: Jika item tidak ditemukan, perbarui indeks hash ke slot berikutnya dalam tabel hash. Jika indeks hash melebihi ukuran tabel hash, kembali ke awal tabel hash.

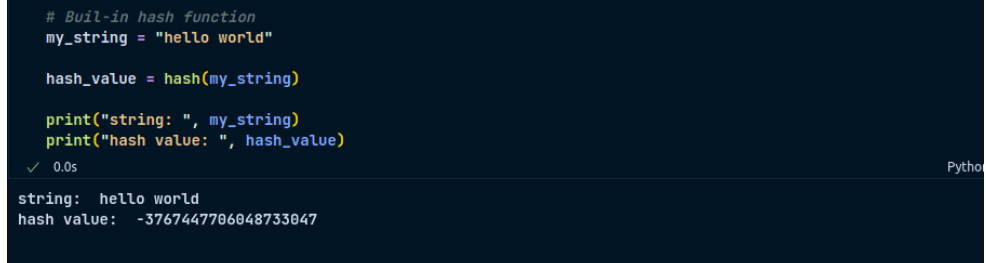
- `return None`: Jika item tidak ditemukan setelah mencari seluruh tabel hash, kembalikan None.
- `def insert(key, data)::` Mendefinisikan fungsi insert yang memasukkan item baru ke dalam tabel hash.
- `item = DataItem(key, data)`: Membuat item baru dengan kunci dan data yang diberikan.
- `hashIndex = hashCode(key)`: Menghitung indeks hash dari kunci menggunakan fungsi hashCode.
- `while hashArray[hashIndex] != None and hashArray[hashIndex].key != -1::` Melakukan loop selama slot tabel hash pada indeks hash tidak None atau tidak berisi item dummy.
- `hashIndex = (hashIndex + 1) % ukuran`: Jika slot tabel hash sudah diisi, perbarui indeks hash ke slot berikutnya dalam tabel hash. Jika indeks hash melebihi ukuran tabel hash, kembali ke awal tabel hash.
- `hashArray[hashIndex] = item`: Menyimpan item baru ke dalam slot tabel hash pada indeks hash.
- `def deleteItem(item)::` Mendefinisikan fungsi deleteItem yang menghapus item dari tabel hash.
- `key = item.key`: Mengambil kunci dari item yang akan dihapus.
- `hashIndex = hashCode(key)`: Menghitung indeks hash dari kunci menggunakan fungsi hashCode.
- `while hashArray[hashIndex] != None::` Melakukan loop selama slot tabel hash pada indeks hash tidak None.
- `if hashArray[hashIndex].key == key::` Jika kunci item pada slot tabel hash sama dengan kunci item yang akan dihapus, maka item tersebut ditemukan.
- `temp = hashArray[hashIndex]`: Menyimpan item yang akan dihapus ke dalam variabel temp.
- `hashArray[hashIndex] = dummyItem`: Mengganti item yang dihapus dengan item dummy.
- `return temp`: Mengembalikan item yang dihapus.
- `hashIndex = (hashIndex + 1) % ukuran`: Jika item tidak ditemukan, perbarui indeks hash ke slot berikutnya dalam tabel hash. Jika

indeks hash melebihi ukuran tabel hash, kembali ke awal tabel hash.

- return None: Jika item tidak ditemukan setelah mencari seluruh tabel hash, kembalikan None.
- def display(): Mendefinisikan fungsi display yang mencetak isi tabel hash.
- for i in range(ukuran): Melakukan loop sebanyak ukuran tabel hash.
- if hashArray[i] != None: Jika slot tabel hash pada indeks i tidak None, cetak item pada slot tersebut.
- print("{}, {}".format(hashArray[i].key, hashArray[i].data), end=" "): Mencetak kunci dan data item.
- else: Jika slot tabel hash pada indeks i None, cetak --.
- print(): Mencetak baris baru setelah mencetak semua item dalam tabel hash.
- if __name__ == "__main__": Memeriksa apakah skrip ini dijalankan sebagai skrip utama.
- insert(1, 20), ..., insert(37, 97): Memasukkan beberapa item ke dalam tabel hash.
- print("insertion done"): Mencetak pesan bahwa semua item telah dimasukkan.
- print("hash table content: "): Mencetak pesan bahwa isi tabel hash akan dicetak.
- display(): Mencetak isi tabel hash.
- item = search(37): Mencari item dengan kunci 37 dalam tabel hash dan menyimpannya ke dalam variabel item.
- if item != None: Jika item ditemukan, cetak data item.
- else: Jika item tidak ditemukan, cetak pesan bahwa item tidak ditemukan.
- deleteItem(item): Menghapus item dari tabel hash.
- item = search(37): Mencari item dengan kunci 37 dalam tabel hash dan menyimpannya ke dalam variabel item.
- if item != None: Jika item ditemukan, cetak data item.

- else:: Jika item tidak ditemukan, cetak pesan bahwa item tidak ditemukan.

Built-in Hash Function



```
# Built-in hash function
my_string = "hello world"

hash_value = hash(my_string)

print("string: ", my_string)
print("hash value: ", hash_value)

✓ 0.0s Python

string: hello world
hash value: -3767447706048733047
```

(Gambar 3.2)

- `my_string = "hello world"`: Mendefinisikan sebuah string "hello world" dan menyimpannya dalam variabel `my_string`.
- `hash_value = hash(my_string)`: Menggunakan fungsi bawaan Python `hash()` untuk menghasilkan nilai hash dari string yang disimpan dalam `my_string`. Nilai hash ini kemudian disimpan dalam variabel `hash_value`.
- `print("string: ", my_string)`: Mencetak string asli ke konsol.
- `print("hash value: ", hash_value)`: Mencetak nilai hash dari string ke konsol.

Hashlib MD5



```
# hashlib MD5
import hashlib

text = "hello world"
hash_object = hashlib.md5(text.encode())
print(hash_object.hexdigest())

✓ 0.0s

5eb63bbbe01eeed093cb22bb8f5acdc3
```

(Gambar 3.3)

- `import hashlib`: Mengimpor modul `hashlib`. Modul ini

mengimplementasikan antarmuka umum untuk banyak algoritma hashing yang berbeda seperti MD5, SHA1, dan lainnya.

- `text = "hello world"`: Mendefinisikan sebuah string "hello world" dan menyimpannya dalam variabel `text`.
- `hash_object = hashlib.md5(text.encode())`: Menggunakan fungsi `md5()` dari modul `hashlib` untuk menghasilkan objek hash dari string yang disimpan dalam `text`. Fungsi `encode()` digunakan untuk mengubah string menjadi bytes, karena fungsi `md5()` membutuhkan input dalam bentuk bytes.
- `print(hash_object.hexdigest())`: Menggunakan metode `hexdigest()` pada objek hash untuk mengembalikan representasi heksadesimal dari hash. Kemudian mencetak hasilnya ke konsol.

Hashlib SHA-1



```
# hashlib SHA-1
import hashlib

text = "hello world"
hash_object = hashlib.sha1(text.encode())
print(hash_object.hexdigest())
```

✓ 0.0s

2aae6c35c94fcfb415dbe95f408b9ce91ee846ed

(Gambar 3.4)

- `import hashlib`: Mengimpor modul `hashlib`.
- `text = "hello world"`: Mendefinisikan sebuah string "hello world" dan menyimpannya dalam variabel `text`.
- `hash_object = hashlib.sha1(text.encode())`: Menggunakan fungsi `sha1()` dari modul `hashlib` untuk menghasilkan objek hash dari string yang disimpan dalam `text`.
- `print(hash_object.hexdigest())`: Menggunakan metode `hexdigest()` pada objek hash untuk mengembalikan representasi heksadesimal dari hash. Kemudian mencetak hasilnya ke konsol.

Hashlib SHA-256

```
# hashlib SHA-256
import hashlib

text = "hello world"
hash_object = hashlib.sha256(text.encode())
print(hash_object.hexdigest())
```

✓ 0.0s

b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9

(Gambar 3.5)

- `import hashlib`: Mengimpor modul hashlib.
- `text = "hello world"`: Mendefinisikan sebuah string "hello world" dan menyimpannya dalam variabel text.
- `hash_object = hashlib.sha256(text.encode())`: Menggunakan fungsi `sha256()` dari modul hashlib untuk menghasilkan objek hash dari string yang disimpan dalam text. Fungsi `encode()` digunakan untuk mengubah string menjadi bytes, karena fungsi `sha256()` membutuhkan input dalam bentuk bytes.
- `print(hash_object.hexdigest())`: Menggunakan metode `hexdigest()` pada objek hash untuk mengembalikan representasi heksadesimal dari hash. Kemudian mencetak hasilnya ke konsol.

Hashlib SHA-384

```
# hashlib SHA-384
import hashlib

text = "hello world"
hash_object = hashlib.sha384(text.encode())
print(hash_object.hexdigest())
```

✓ 0.0s

fdbd8e75a67f29f701a4e040385e2e23986303ea10239211af907fcbb83578b3e417cb71ce646efd0819dd8c088de1bd

(Gambar 3.6)

- `import hashlib`: Mengimpor modul hashlib.
- `text = "hello world"`: Mendefinisikan sebuah string "hello world" dan menyimpannya dalam variabel text.
- `hash_object = hashlib.sha384(text.encode())`: Menggunakan fungsi `sha384()` dari modul hashlib untuk menghasilkan objek hash dari string yang disimpan dalam text. Fungsi `encode()` digunakan untuk mengubah string menjadi bytes, karena fungsi `sha384()` membutuhkan input dalam bentuk bytes.
- `print(hash_object.hexdigest())`: Menggunakan metode `hexdigest()` pada objek hash untuk mengembalikan representasi heksadesimal dari hash. Kemudian mencetak hasilnya ke konsol.

Hashlib SHA-512



```
# hashlib SHA-512
import hashlib

text = "hello world"
hash_object = hashlib.sha512(text.encode())
print(hash_object.hexdigest())
```

✓ 0.0s

309ecc489c12d6eb4cc40f50c902f2b4d0ed77ee511a7c7a9bcd3ca86d4cd86f989dd35bc5ff499670da34255b45b0cfd

(Gambar 3.7)

- `import hashlib`: Mengimpor modul hashlib.
- `text = "hello world"`: Mendefinisikan sebuah string "hello world" dan menyimpannya dalam variabel text.
- `hash_object = hashlib.sha512(text.encode())`: Menggunakan fungsi `sha512()` dari modul hashlib untuk menghasilkan objek hash dari string yang disimpan dalam text. Fungsi `encode()` digunakan untuk mengubah string menjadi bytes, karena fungsi `sha512()` membutuhkan input dalam bentuk bytes.
- `print(hash_object.hexdigest())`: Menggunakan metode `hexdigest()` pada objek hash untuk mengembalikan representasi heksadesimal dari hash. Kemudian mencetak hasilnya ke konsol.

Seperate Chaining

```
class node:
    def __init__(self, key, value):
        self.key = key
        self.data = value
        self.next = None

class separateChainingHashTable:
    def __init__(self, size):
        self.size = size
        self.table = [None] * size

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

    def insert(self, key, value):
        index = self.hashFunction(key)
        if self.table[index] == None:
            self.table[index] = node(key, value)
        else:
            current = self.table[index]
            while current.next != None:
                current = current.next
            current.next = node(key, value)

    def search(self, key):
        index = self.hashFunction(key)
        current = self.table[index]
        while current != None:
            if current.key == key:
                return current.data
            current = current.next
        return None

# contoh penggunaan
hashTable = separateChainingHashTable(10)
hashTable.insert("apple", 10)
hashTable.insert("banana", 20)
hashTable.insert("orange", 30)

print(hashTable.search("apple"))
print(hashTable.search("banana"))
print(hashTable.search("orange"))
```

(Gambar 3.8)

- class node: Mendefinisikan kelas node yang akan digunakan sebagai elemen dalam rantai hash. Setiap node memiliki key, data, dan next yang merujuk ke node berikutnya dalam rantai.
- class separateChainingHashTable: Mendefinisikan kelas separateChainingHashTable yang akan digunakan sebagai struktur data hash table dengan metode separate chaining.
- def __init__(self, size): Konstruktor untuk kelas separateChainingHashTable. Menerima ukuran tabel hash dan menginisialisasi tabel dengan None.
- def hashFunction(self, key): Fungsi hash yang digunakan untuk menghitung indeks dalam tabel hash.
- def insert(self, key, value): Metode untuk memasukkan pasangan kunci-nilai ke dalam tabel hash.
- def search(self, key): Metode untuk mencari nilai berdasarkan kunci dalam tabel hash.
- hashTable = separateChainingHashTable(10): Membuat objek hashTable dari kelas separateChainingHashTable dengan ukuran 10.

- `hashTable.insert("apple", 10), hashTable.insert("banana", 20), hashTable.insert("orange", 30)`: Memasukkan beberapa pasangan kunci-nilai ke dalam `hashTable`.
- `print(hashTable.search("apple")), print(hashTable.search("banana")), print(hashTable.search("orange"))`: Mencetak nilai yang dikembalikan oleh metode `search()` untuk beberapa kunci.

Open Addressing

```
# Open Addressing
class openAddressHashTable:
    def __init__(self, size):
        self.size = size
        self.table = [None] * size

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

    def insert(self, key, value):
        index = self.hashFunction(key)
        while self.table[index] != None:
            index = (index + 1) % self.size
        self.table[index] = node(key, value)

    def search(self, key):
        index = self.hashFunction(key)
        while self.table[index] != None:
            if self.table[index].key == key:
                return self.table[index].data
            index = (index + 1) % self.size
        return None

# contoh penggunaan
hashTable = openAddressHashTable(10)
hashTable.insert("apple", 10)
hashTable.insert("banana", 20)
hashTable.insert("orange", 30)

print(hashTable.search("apple"))
print(hashTable.search("banana"))
print(hashTable.search("orange"))
```

✓ 0.0s Python

10
20
30

(Gambar 3.9)

- `class openAddressHashTable:` Mendefinisikan kelas `openAddressHashTable` yang akan digunakan sebagai struktur data hash table dengan metode open addressing.
- `def __init__(self, size):` Konstruktor untuk kelas `openAddressHashTable`. Menerima ukuran tabel hash dan menginisialisasi tabel dengan `None`.
- `def hashFunction(self, key):` Fungsi hash yang digunakan untuk

menghitung indeks dalam tabel hash.

- `def insert(self, key, value)`: Metode untuk memasukkan pasangan kunci-nilai ke dalam tabel hash. Jika terjadi tabrakan, indeks ditingkatkan sampai ditemukan slot kosong.
- `def search(self, key)`: Metode untuk mencari nilai berdasarkan kunci dalam tabel hash. Jika kunci tidak ditemukan, metode ini akan terus mencari sampai menemukan slot kosong.
- `hashTable = openAddressHashTable(10)`: Membuat objek `hashTable` dari kelas `openAddressHashTable` dengan ukuran 10.
- `hashTable.insert("apple", 10), hashTable.insert("banana", 20), hashTable.insert("orange", 30)`: Memasukkan beberapa pasangan kunci-nilai ke dalam `hashTable`.
- `print(hashTable.search("apple")), print(hashTable.search("banana")), print(hashTable.search("orange"))`: Mencetak nilai yang dikembalikan oleh metode `search()` untuk beberapa kunci.

- **Latihan**



```
# Latihan

def hash_sort(arr):
    # Langkah 1: Buat array hash dengan ukuran (max_element)
    max_element = max(arr)
    hash_array = [0] * (max_element + 1)

    # Langkah 2: Telusuri semua elemen dan hitung jumlah kemunculan elemen tertentu
    for num in arr:
        hash_array[num] += 1

    # Langkah 3 dan 4: Lakukan iterasi dari 0 hingga max_element dalam array hash
    # Jika nilai yang disimpan pada posisi hash mana pun lebih dari 0, cetak elemen tersebut
    sorted_arr = []
    for i in range(len(hash_array)):
        if hash_array[i] > 0:
            # Langkah 5: Hash[i] memiliki hitungan berapa kali suatu elemen ada dalam daftar
            # Jadi ketika >0, kami mencetak berapa kali elemen tersebut
            for _ in range(hash_array[i]):
                sorted_arr.append(i)

    return sorted_arr

# Masukan
arr = [8, 2, 4, 6, 7, 1]

# Keluaran
print(hash_sort(arr)) # Output: [1, 2, 4, 6, 7, 8]
```

✓ 0.0s Python

[1, 2, 4, 6, 7, 8]

(Gambar 4.1)

- `max_element = max(arr)`: Mencari elemen maksimum dalam array input. Ini akan digunakan untuk menentukan ukuran array hash.
- `hash_array = [0] * (max_element + 1)`: Membuat array hash dengan ukuran `max_element + 1`, diinisialisasi dengan nol. Array ini akan digunakan untuk menghitung kemunculan setiap elemen dalam array input.
- Loop `for num in arr`: berfungsi untuk mengiterasi setiap angka dalam array input. Untuk setiap angka, ia menambahkan indeks yang sesuai dalam array hash.
- Loop `for i in range(len(hash_array))`: berfungsi untuk mengiterasi setiap indeks dalam array hash. Jika nilai pada indeks lebih dari nol, berarti angka yang sesuai dengan indeks tersebut ada dalam array input.
- Loop dalam `for _ in range(hash_array[i])`: berjalan sebanyak hitungan angka dalam array input. Setiap kali, ia menambahkan angka ke list `sorted_arr`.
- Akhirnya, fungsi mengembalikan list `sorted_arr`, yang merupakan versi yang sudah diurutkan dari array input.

- **Kesimpulan**

Hashing adalah prinsip dasar dalam struktur data yang memungkinkan penyimpanan dan pengambilan data dengan efisien serta akses yang cepat. Ini dilakukan dengan cara memetakan data ke indeks tertentu dalam tabel hash menggunakan fungsi hash. Fungsi hash ini memungkinkan pengambilan informasi secara cepat berdasarkan kunci yang diberikan.

Hashing merupakan teknik yang sering digunakan dalam berbagai aplikasi pemrograman, termasuk basis data dan sistem caching, untuk meningkatkan efisiensi dalam pencarian dan pengambilan data.

Dengan menggunakan teknik hashing, pencarian dan pengambilan data menjadi lebih efisien karena prosesnya dapat dilakukan dengan cepat berdasarkan indeks yang dihasilkan oleh fungsi hash, tanpa perlu melakukan pencarian linier pada seluruh data.