

LAPORAN PRAKTIKUM STRUKTUR DATA

MODUL KE-09

HASHING DALAM PYTHON



Disusun Oleh:

Nama : Restu Wibisono
NPM : 2340506061
Kelas : 03 (Tiga)

Program Studi S1 Teknologi Informasi

Fakultas Teknik, Universitas Tidar

Genap 2023/2024

I. Tujuan Praktikum

Adapun tujuan praktikum ini sebagai berikut :

1. Mahasiswa mampu menerapkan konsep Hashing pada bahasa pemrograman python

II. Dasar Teori

Peningkatan kebutuhan akan struktur data Hash sejalan dengan pertumbuhan yang cepat dari jumlah data di internet. Meskipun ukuran data dalam kegiatan pemrograman sehari-hari mungkin tidak sebesar itu, namun tetap memerlukan penyimpanan, akses, dan pemrosesan yang efisien. Meskipun Struktur data Array umum digunakan, namun memiliki batasan efisiensi karena walaupun penyimpanannya cepat ($O(1)$), pencariannya memerlukan waktu logaritmik ($O(\log n)$), yang mungkin tidak efisien terutama untuk data yang besar.

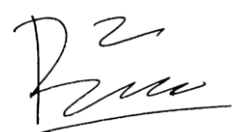
Dengan demikian, diperlukan sebuah struktur data baru yang mampu menyimpan dan mencari data dengan cepat dalam waktu konstan ($O(1)$). Inilah alasan mengapa struktur data Hashing sangat penting. Dengan menggunakan Hashing, data dapat disimpan dan diambil dengan mudah serta efisien dalam waktu yang konstan.

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:

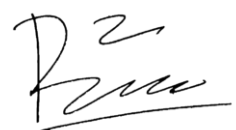
- **Kunci:** Sebuah nilai yang bisa berupa string atau bilangan bulat yang digunakan sebagai input dalam fungsi hash untuk menentukan indeks atau lokasi penyimpanan dalam struktur data.

Tanda Tangan



- 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.

Tanda Tangan

A handwritten signature in black ink, consisting of a stylized 'D' followed by a series of loops and a horizontal stroke at the bottom.

III. Hasil dan Pembahasan

1. Tabel 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
```

(Gambar 3.1.1)

```
# 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()

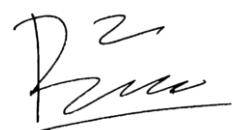
if __name__ == "__main__":
    insert(1, 20)
    insert(2, 70)
    insert(42, 80)
    insert(4, 25)
    insert(12, 44)
    insert(14, 32)
    insert(17, 11)
    insert(13, 78)
    insert(37, 97)
    print("insertion done")
    print("hash table content: ")
    display()
    item = search(37)
    if item != 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")
```

Python

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

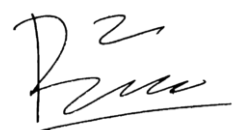
(Gambar 3.1.2)

Tanda Tangan



1. Program menetapkan ukuran tabel hash menjadi 20. Tabel hash digunakan untuk menyimpan data menggunakan kunci tertentu.
2. Sebuah kelas bernama 'DataItem' didefinisikan di sini. Kelas ini digunakan untuk menyimpan pasangan kunci-nilai tabel hash.
3. Fungsi konstruktor untuk kelas DataItem adalah 'def __init__(self, key, data)'. Fungsi ini dijalankan saat membuat objek baru kelas 'DataItem'.
4. 'self.key = key' dan 'self.data = data': Baris ini mengatur atribut kunci dan data dari objek 'DataItem' sesuai dengan nilai yang diberikan ketika objek dibuat.
5. Sebuah array dibuat dengan panjang sesuai dengan nilai 'ukuran'. Array ini akan digunakan sebagai tabel hash untuk menyimpan data.
6. Program menciptakan sebuah objek 'dummyItem = DataItem(-1, -1)' disebut sebagai item palsu (dummy item). Objek ini akan digunakan untuk menandai slot yang telah dihapus dari tabel hash.
7. Variabel 'item' digunakan untuk menyimpan item yang sedang dicari atau dihapus dari tabel hash. Nilainya diatur sebagai 'None' untuk menandakan bahwa belum ada item yang disimpan di dalamnya.
8. Fungsi 'def hashCode(key)': mendefinisikan cara untuk menghitung kode hash dari suatu kunci. Kode hash ini akan digunakan untuk menentukan di mana data akan disimpan dalam tabel hash.
9. Selanjutnya 'return key % ukuran': mengembalikan sisa pembagian kunci dengan nilai 'ukuran' tabel hash. Ini metode hashing sederhana digunakan dalam contoh ini.
10. Fungsi 'search(key)' digunakan saat kita ingin mencari item dengan kunci tertentu di dalam tabel hash.

Tanda Tangan



11. Pertama, program menghitung indeks hash dari kunci yang diberikan menggunakan fungsi 'hashCode'.
12. Kemudian, program memulai proses pencarian dengan memeriksa apakah ada data di indeks hash tersebut. Jika ada, program akan masuk ke dalam loop untuk mengecek item di slot tersebut.
13. Jika setelah mencari seluruh tabel hash tidak ditemukan item dengan kunci yang dicari, maka program akan mengembalikan nilai 'None'.

2. Buil-in hash Function

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

hash_value = hash(my_string)

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

Python

string: hello world
hash value: -3767447706048733047

(Gambar 3.2)

1. Sebuah string "hello world" didefinisikan dan disimpan dalam variabel 'my_string'.
2. Fungsi bawaan Python 'hash()' digunakan untuk menghasilkan nilai hash dari string yang disimpan dalam 'my_string'. Nilai hash ini kemudian disimpan dalam variabel 'hash_value'.
3. String asli dicetak ke konsol menggunakan perintah 'print("string: ", my_string)'.
4. Nilai hash dari string dicetak ke konsol menggunakan perintah 'print("hash value: ", hash_value)'.

3. Hashlib MD5

```
# hashlib MD5
import hashlib

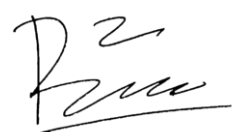
text = "hello world"
hash_object = hashlib.md5(text.encode())
print(hash_object.hexdigest())
```

Python

5eb63bbbe01eeed093cb22bb8f5acdc3

(Gambar 3.3)

Tanda Tangan



1. 'Modul 'hashlib' diimpor untuk digunakan. Modul ini menyediakan berbagai algoritma hashing seperti MD5, SHA1, dan lainnya.
2. 'Sebuah string "hello world" didefinisikan dan disimpan dalam variabel 'text'.
3. 'Fungsi 'md5()' dari modul 'hashlib' digunakan untuk menghasilkan objek hash dari string yang disimpan dalam 'text'. Untuk menggunakan fungsi 'md5()', string perlu diubah menjadi bytes menggunakan fungsi 'encode()'.
4. 'Metode 'hexdigest()' dipanggil pada objek hash untuk mengembalikan representasi heksadesimal dari hash. Representasi ini kemudian dicetak ke konsol.

4. Hashlib SHA-1

```
# hashlib SHA-1
import hashlib

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

2aae6c35c94fcfb415dbe95f408b9ce91ee846ed

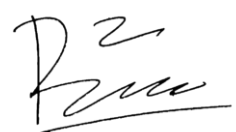
Python

(Gambar 3.4)

1. 'Modul 'hashlib' diimpor untuk digunakan dalam program.
2. 'Sebuah string "hello world" didefinisikan dan disimpan dalam variabel 'text'.
3. 'Fungsi 'sha1()' dari modul 'hashlib' digunakan untuk menghasilkan objek hash dari string yang disimpan dalam 'text'.
4. 'Metode 'hexdigest()' dipanggil pada objek hash untuk mengembalikan representasi heksadesimal dari hash. Hasilnya kemudian dicetak ke konsol.

5. Hashlib SHA-256

Tanda Tangan



```
# hashlib SHA-256
import hashlib

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

Python

b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9

(Gambar 3.5)

1. 'Modul 'hashlib' diimpor untuk digunakan dalam program.
2. 'Sebuah string "hello world" didefinisikan dan disimpan dalam variabel 'text'.
3. 'Fungsi 'sha256()' dari modul 'hashlib' digunakan 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.
4. 'Metode 'hexdigest()' dipanggil pada objek hash untuk mengembalikan representasi heksadesimal dari hash. Hasilnya kemudian dicetak ke konsol.

6. Hashlib SHA-384

```
# hashlib SHA-384
import hashlib

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

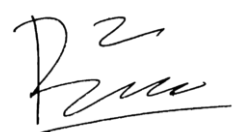
Python

fdbd8e75a67f29f701a4e040385e2e23986303ea10239211af907fcbb83578b3e417cb71ce646efd0819dd8c0

(Gambar 3.6)

1. 'Modul 'hashlib' diimpor untuk digunakan dalam program.
2. 'Sebuah string "hello world" didefinisikan dan disimpan dalam variabel 'text'.
3. 'Fungsi 'sha384()' dari modul 'hashlib' digunakan 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.
4. 'Metode 'hexdigest()' dipanggil pada objek hash untuk mengembalikan representasi heksadesimal dari hash. Hasilnya kemudian dicetak ke konsol.

Tanda Tangan



7. Hashlib SHA-512

```
# hashlib SHA-512
import hashlib

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

Python

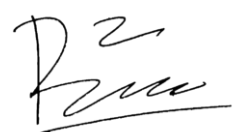
309ecc489c12d6eb4cc40f50c902f2b4d0ed77ee511a7c7a9bcd3ca86d4cd86f989dd35bc5ff499670da34255

(Gambar 3.7)

1. 'Modul 'hashlib' diimpor untuk digunakan dalam program.
2. 'Sebuah string "hello world" didefinisikan dan disimpan dalam variabel 'text'.
3. 'Fungsi 'sha512()' dari modul 'hashlib' digunakan 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.
4. 'Metode 'hexdigest()' dipanggil pada objek hash untuk mengembalikan representasi heksadesimal dari hash. Hasilnya kemudian dicetak ke konsol.

8. Seprate Chaning

Tanda Tangan



```

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"))

```

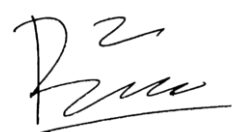
10
20
30

Python

(Gambar 3.8)

1. 'Kelas 'node' didefinisikan untuk digunakan sebagai elemen dalam rantai hash. Setiap node memiliki atribut key, data, dan next yang merujuk ke node berikutnya dalam rantai.
2. 'Kelas 'separateChainingHashTable' didefinisikan sebagai struktur data hash table dengan metode separate chaining.
3. 'Konstruktor '.__init__(self, size)' untuk kelas 'separateChainingHashTable' digunakan untuk menginisialisasi tabel hash dengan ukuran yang ditentukan. Awalnya, tabel diinisialisasi dengan nilai None.
4. 'Metode 'hashFunction(self, key)' digunakan untuk menghitung indeks dalam tabel hash berdasarkan kunci yang diberikan.

Tanda Tangan



5. 'Metode 'insert(self, key, value)' digunakan untuk memasukkan pasangan kunci-nilai ke dalam tabel hash.
6. 'Metode 'search(self, key)' digunakan untuk mencari nilai berdasarkan kunci dalam tabel hash.
7. 'Objek 'hashTable' dibuat dari kelas 'separateChainingHashTable' dengan ukuran 10.
8. 'Beberapa pasangan kunci-nilai dimasukkan ke dalam 'hashTable' menggunakan metode 'insert("apple", 10)', 'insert("banana", 20)', dan 'insert("orange", 30)'.
9. ' Nilai yang dikembalikan oleh metode 'search()' untuk beberapa kunci, seperti "apple", "banana", dan "orange", dicetak menggunakan perintah 'print(hashTable.search("apple"))', 'print(hashTable.search("banana"))', dan 'print(hashTable.search("orange"))'.

9. 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

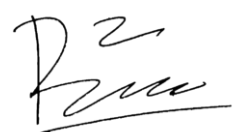
# contoh penggunaan
hashTable = openAddressHashTable(10)
hashTable.insert("apple", 10)
hashTable.insert("banana", 20)
hashTable.insert("orange", 30)

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

Python

10
20
30

Tanda Tangan



(Gambar 3.9)

1. Kelas 'openAddressHashTable' didefinisikan sebagai struktur data hash table dengan metode open addressing.
2. Konstruktor '`__init__(self, size)`' untuk kelas 'openAddressHashTable' digunakan untuk menginisialisasi tabel hash dengan ukuran yang ditentukan. Pada awalnya, tabel diinisialisasi dengan nilai None.
3. Metode '`hashFunction(self, key)`' digunakan untuk menghitung indeks dalam tabel hash berdasarkan kunci yang diberikan.
4. Metode '`insert(self, key, value)`' digunakan untuk memasukkan pasangan kunci-nilai ke dalam tabel hash. Jika terjadi tabrakan, indeks akan ditingkatkan sampai ditemukan slot kosong.
5. Metode '`search(self, key)`' digunakan untuk mencari nilai berdasarkan kunci dalam tabel hash. Jika kunci tidak ditemukan, metode ini akan terus mencari sampai menemukan slot kosong.
6. Objek 'hashTable' dibuat dari kelas 'openAddressHashTable' dengan ukuran 10.
7. Beberapa pasangan kunci-nilai dimasukkan ke dalam 'hashTable' menggunakan metode '`insert("apple", 10)`', '`insert("banana", 20)`', dan '`insert("orange", 30)`'.
8. Nilai yang dikembalikan oleh metode '`search()`' untuk beberapa kunci, seperti "apple", "banana", dan "orange", dicetak menggunakan perintah '`print(hashTable.search("apple"))`', '`print(hashTable.search("banana"))`', dan '`print(hashTable.search("orange"))`'.

Tanda Tangan



IV. Latihan

```
# Latihan

def hash_sort(arr):
    max_element = max(arr)
    hash_array = [0] * (max_element + 1)

    for num in arr:
        hash_array[num] += 1

    sorted_arr = []
    for i in range(len(hash_array)):
        if hash_array[i] > 0:
            for _ in range(hash_array[i]):
                sorted_arr.append(i)

    return sorted_arr

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

print(hash_sort(arr))
```

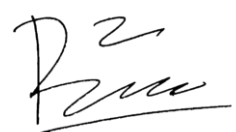
✓ 0.0s Python

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

(Gambar 4.1)

1. Menggunakan 'max(arr)' untuk menemukan elemen terbesar dalam array input guna menetapkan ukuran array hash.
2. Array hash dibuat dengan ukuran 'max_element + 1' dan diinisialisasi dengan nol. Array ini berperan dalam menghitung kemunculan setiap elemen dalam array input.
3. Dengan menggunakan loop 'for num in arr', setiap angka dalam array input diiterasi. Pada setiap iterasi, indeks yang sesuai dalam array hash akan ditambahkan.
4. Melalui loop 'for i in range(len(hash_array))', setiap indeks dalam array hash diperiksa. Jika nilai pada indeks tersebut lebih besar dari nol, berarti angka yang sesuai dengan indeks tersebut ada dalam array input.
5. Dalam loop bersarang 'for _ in range(hash_array[i])', iterasi dilakukan sebanyak kemunculan angka pada indeks tertentu dalam array hash. Pada setiap iterasi, angka ditambahkan ke dalam list 'sorted_arr'.
6. Sebagai hasil akhir, fungsi mengembalikan list 'sorted_arr', yang merupakan array input yang telah diurutkan.

Tanda Tangan



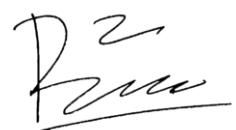
V. Kesimpulan

Hashing adalah prinsip dasar struktur data yang menyediakan akses yang efektif dan cepat ke penyimpanan dan pencarian data. Ini dilakukan dengan menggunakan fungsi hash untuk menampilkan data dengan indeks spesifik dalam tabel hash. Fungsi hash ini memungkinkan Anda mengambil informasi dengan cepat berdasarkan kunci tertentu.

Hashing adalah teknik yang biasa digunakan dalam berbagai aplikasi pemrograman, seperti database dan sistem caching, untuk meningkatkan efisiensi pencarian dan pengambilan data.

Teknik hashing membuat pencarian dan pengambilan data menjadi lebih efisien karena tidak perlu melakukan pencarian linier atas semua data dan prosesnya dapat diselesaikan dengan cepat berdasarkan indeks yang dibuat oleh fungsi hashing.

Tanda Tangan

A handwritten signature in black ink, consisting of stylized cursive letters, enclosed within a rectangular box.