

Laporan Praktikum Struktur Data

Semester Gasal 2021/2022



NIM	71200634
Nama Lengkap	Egi Granaldi Ginting
Minggu ke / Materi	12 / Hash Table

SAYA MENYATAKAN BAHWA LAPORAN PRAKTIKUM INI SAYA BUAT DENGAN USAHA SENDIRI TANPA MENGGUNAKAN BANTUAN ORANG LAIN. SEMUA MATERI YANG SAYA AMBIL DARI SUMBER LAIN SUDAH SAYA CANTUMKAN SUMBERNYA DAN TELAH SAYA TULIS ULANG DENGAN BAHASA SAYA SENDIRI.

SAYA SANGGUP MENERIMA SANKSI JIKA MELAKUKAN KEGIATAN PLAGIASI, TERMASUK SANKSI TIDAK LULUS MATA KULIAH INI.

PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS KRISTEN DUTA WACANA
YOGYAKARTA
2021

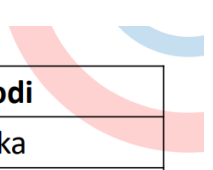
BAGIAN 1: Hash Table

Hashing digunakan untuk menyimpan data yang cukup besar pada ADT yang disebut hash table. Ukuran Hash table (H-size), biasanya lebih besar dari jumlah data yang hendak disimpan. Fungsi dari hash adalah memetakan elemen pada indeks dari hash table. Hashing adalah teknik untuk melakukan penambahan, penghapusan dan pencarian dengan Constant Average Time. Hash Table adalah array dengan sel – sel yang ukurannya telah ditentukan dan dapat berisi data atau key yang berkesesuaian dengan data.

Pengaksesan data pada dictionary sangat cepat, karena implementasinya berdasarkan hash table. Nilai dari data disimpan dalam list / array yang dapat diakses menggunakan index-nya. Linear search dan Binary search melakukan pencarian karena tidak mengetahui index dari data yang dicari. Pengaksesan data akan lebih cepat jika kita mengetahui index-nya. Untuk mengetahui index dari data yang ingin diambil caranya adalah sebagai berikut:

- Membutuhkan Key dan Fungsi Hash
- Masukkan key ke dalam fungsi hash untuk mendapatkan index dari data yang ingin dicari
- Lokasi dari data didapatkan dari hasil fungsi hash dari nilai data tersebut.

Fungsi Hash



Kode	Nama Prodi
71	Informatika
72	Sistem Informasi
41	Kedokteran
11	Manajemen

$$F(\text{key}) = \text{key} \bmod 7$$

Maka untuk key 71, lokasi penyimpanannya dapat dihitung dengan fungsi hash tersebut, yaitu:
 $F(71) = 71 \bmod 7 = 1$

Berarti Informatika disimpan dalam index ke-1

Fungsi hash harus memiliki sifat berikut :

- Mudah Dihitung
- Membagi key secara rata pada seluruh sel.
- Sebuah fungsi hash sederhana adalah menggunakan fungsi mod (sisa bagi) dengan bilangan prima.
- Fungsi yang dapat memetakan suatu nilai data (yang panjang/besarnya bisa bervariasi) ke dalam suatu nilai yang memiliki rentang yang tetap (fixed, sudah ditentukan sebelumnya)
 - Merupakan fungsi satu arah. Dari key bisa didapatkan value, tetapi tidak sebaliknya
- Fungsi Hash yang bagus:
 - Proses perhitungannya cepat
 - Deterministic = untuk key yang sama, maka nilai hash yang dihasilkan harus sama
 - Fixed-length value = hasilnya adalah nilai dalam rentang yang telah ditentukan
 - Evenly distribution = menyebar key secara merata (tidak mengumpul di satu tempat saja)
- Perfect hash function
 - Hanya terjadi jika seluruh key sudah diketahui sebelumnya

Permasalahan Fungsi Hash

- Simple Uniform Hashing
 - Setiap key memiliki peluang yang sama untuk menempati salah satu dari slot yang tersedia di dalam hash table
 - Hasil dari hashing tidak mengumpul di satu area saja
- Jika $f(\text{key}) = \text{key} \bmod 7$, maka
 - Key = 9, 16, 37, 2 => semuanya akan menghasilkan index ke-2
- Karena memetakan nilai yang panjang/besarnya bervariasi ke tempat yang ukurannya sudah ditetapkan, maka ada kemungkinan fungsi hash menghasilkan nilai yang sama walaupun key yang digunakan berbeda
- Hash Collision merupakan masalah utama yang sering ditemui pada hash table
 - Perlu diperhatikan Time dan ruang space

Time and Space Problem

- Bagaimana jika fungsi hash yang dipakai adalah $f(\text{key}) = \text{key}$?
 - 71 berarti disimpan di index ke-71
 - 72 berarti disimpan di index ke-72
 - 41 berarti disimpan di index ke-41
 - Mengasumsikan memiliki space yang besar (menjadi tidak efisien karena ada banyak tempat kosong yang tidak terpakai)
- Mengatasi Collision
 - Seharusnya dilakukan dengan cepat dan efisien, karena berkaitan dengan masalah waktu.
 - Jika waktu tidak masalah, maka cukup dengan linier probing saja, tetapi membutuhkan jumlah space yang besar

Jenis Fungsi Hash

- Modulo
 - $f(\text{key}) = \text{key} \bmod n$
 - Dimana n biasanya adalah kapasitas maksimal dari hash table
 - Bisa juga n berupa bilangan prima
- Multiplication
 - Pilih suatu nilai A yang memenuhi $0 < A < 1$. Biasanya $A = 0.618$ (golden ratio)
 - $f(\text{key}) = \text{floor}(m * ((\text{key} * A) - \text{floor}(\text{key} * A)))$
 - m adalah nilai yang lebih besar dari kapasitas maksimal hash table (kelipatan 2)
 - floor() adalah fungsi pembulatan ke bawah, untuk menghilangkan pecahan/desimal, menghasilkan bilangan bulat (integer)
- Universal Hashing
 - Menyediakan bermacam – macam fungsi hash, lalu digunakan secara acak.

Hash Pada String

- Menggunakan modulo / multiplication tentu membutuhkan bilangan (real / integer)
 - Menggunakan modulo / multiplication tentu membutuhkan bilangan (real / integer)
 - Biasanya akan menggunakan nilai ASCII / Unicode code point nya
 - Pada python bisa didapatkan dengan fungsi ord() – menghasilkan Unicode code point.
- Posisi karakter pada string sangat penting. Jika berbeda urutan, maka stringnya akan berbeda juga

Contoh Hash Table

Misalkan kita ingin menyimpan data berikut ini pada sebuah hash table berukuran 13:

• 93, 70, 137, 24, 51, 83, 65, 111

Fungsi Hash yang akan dipakai adalah $f(\text{key}) = \text{key} \bmod 13$

- Digunakan mod 13 karena hash table berukuran 13

- Ini contoh yang sangat sederhana, dalam banyak kasus, fungsi hash bisa lebih kompleks lagi perhitungannya.

Contoh Hash Table

$f(\text{key}) = \text{key} \bmod 13$

93, 70, 137, 24, 51, 83, 65, 111

$f(93) = 93 \bmod 13 = 2$

$f(70) = 70 \bmod 13 = 5$

$f(137) = 137 \bmod 13 = 7$

$f(24) = 24 \bmod 13 = 11$

$f(51) = 51 \bmod 13 = 12$

$f(83) = 83 \bmod 13 = 5$

83 →
Collision!

0	
1	
2	93
3	
4	
5	70
6	
7	137
8	
9	
10	
11	24
12	51

Gambar 1 Contoh Hash Table

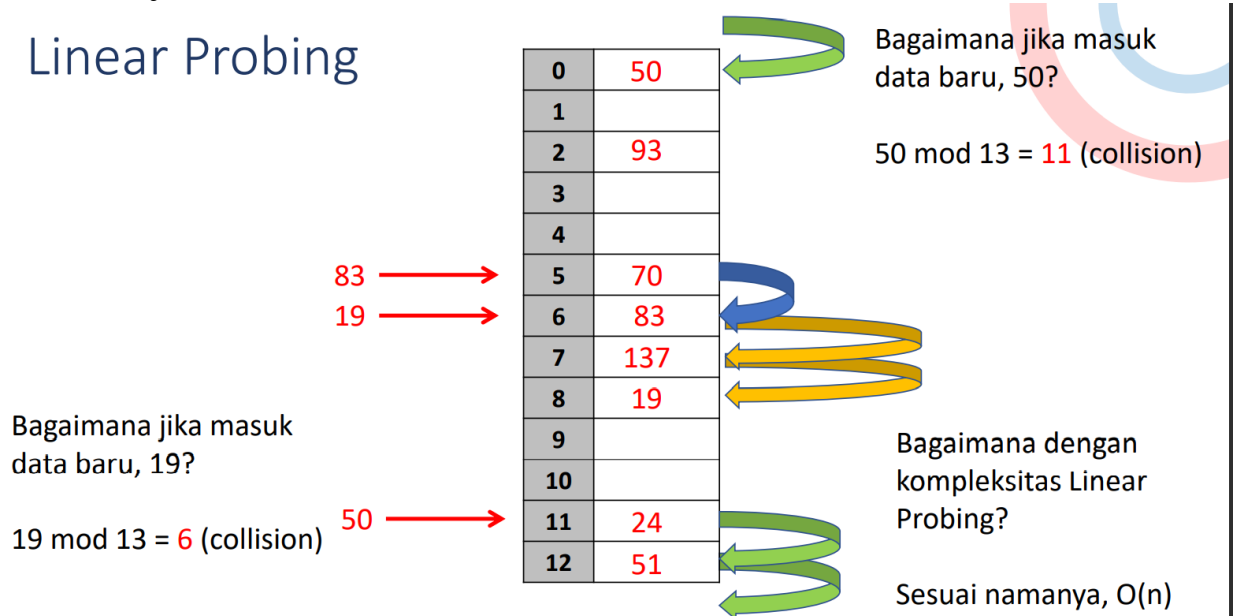
Mengatasi Collision

- Fungsi hash yang bagus seharusnya dapat meminimalkan terjadinya collision
 - Load factor = $100\% \times (\text{jumlah data tersimpan} / \text{kapasitas hash table})$
 - Jika load factor semakin besar, maka kemungkinan terjadinya collision akan semakin besar juga.
- Ada dua strategi umum untuk mengatasi collision :
 - Closed Hashing (Open Addressing) : Linear Probing, Quadratic Probing, Double Hashing
 - Open Hashing : Separate Chaining

Linear Probing

- Menggunakan fungsi linear $f(i) = i$
- Bila terjadi collision, cari posisi pertama pada tabel yang terdekat dengan posisi yang seharusnya.
- Fungsi linear relative paling sederhana
- Linear Probing hanya disarankan untuk ukuran hash table yang ukurannya lebih besar dua kali dari jumlah data.

Linear Probing



Gambar 2 Linear Probing

Quadratic Probing

- Jika linear probing mencari tempat alternatif di sebelahnya (selisih 1), maka quadratic probing mencari tempat alternatif berdasarkan jarak kuadrat
- Menimbulkan banyak permasalahan bila hash table telah terisi lebih dari setengah.
- Dengan ukuran hash table yang merupakan bilangan prima dan hash table yang terisi kurang dari setengah, strategy quadratic probe dapat selalu menemukan lokasi untuk setiap elemen baru.

Double Hashing

- Salah satu syaratnya ukuran hash table haruslah bilangan prima.
- Menyiapkan dua fungsi hash sekaligus
 - Jika fungsi hash pertama menghasilkan collision, maka hitung lokasi baru dengan fungsi hash kedua

- Harus hati-hati dalam memilih fungsi hash kedua untuk menjamin agar tidak menghasilkan nilai 0 dan mem-probe ke seluruh sel.

Open Hashing

Permasalahan Collision diselesaikan dengan menambahkan seluruh elemen yang memilih nilai hash sama pada sebuah set.

Open Hashing menyediakan sebuah linked list untuk setiap elemen yang memiliki nilai hash sama. Tiap sel pada hash berisi pointer ke sebuah linked list yang berisikan data / elemen

Fungsi Open Hashing

- Menambahkan sebuah elemen ke dalam tabel
 - Dilakukan dengan menambahkan elemen pada akhir atau awal linked-list yang sesuai dengan nilai hash.
 - Bergantung apakah perlu ada pengujian nilai duplikasi atau tidak.
 - Dipengaruhi berapa sering elemen terakhir akan diakses
-
- Untuk pencarian, gunakan fungsi hash untuk menentukan linked list mana yang memiliki elemen yang dicari, kemudian lakukan pembacaan terhadap linked list tersebut
 - Penghapusan dilakukan pada linked list setelah pencarian elemen dilakukan.
 - Dapat saja digunakan struktur data lain selain linked list untuk menyimpan elemen yang memiliki fungsi hash yang sama tersebut
 - Kelebihan utama dari metode ini adalah dapat menyimpan data yang tak terbatas. (dynamic expansion).
 - Kekurangan utama adalah penggunaan memory pada tiap sel.

Separate Chaining

Bagaimana jika masuk data baru, 19?


$19 \bmod 13 = 6$ (collision)

Bagaimana jika masuk data baru, 50?

$50 \bmod 13 = 11$ (collision)

Bagaimana jika masuk data baru, 115?

$115 \bmod 13 = 11$ (collision)



0	
1	
2	93
3	
4	
5	70
6	83
7	137
8	
9	
10	
11	24
12	51

Diagram illustrating Separate Chaining. A hash table with 13 slots (0-12) is shown. Slots 2, 5, 6, 7, 11, and 12 contain values 93, 70, 83, 137, 24, and 51 respectively. Arrows indicate collisions: 19 points to slot 6 (83), 50 points to slot 11 (24), and 115 points to slot 11 (24).

Gambar 3 Contoh Open Hashing atau Separate Chaining

Penggunaan Hash Table

- Implementasi Dictionary
- Representasi Object pada bahasa pemrograman dinamis yang mendukung OOP
- Database Indexing
- Caches
- Digunakan dalam berbagai macam algoritma untuk mempercepat penyimpanan dan pencarian data

BAGIAN 2: SOAL UNGUIDED

Pada bagian ini anda menuliskan jawaban dari soal-soal unguided yang diberikan. Jawaban anda harus disertai dengan source code, penjelasan dan screenshot output

Unguided 1

```
class bukuTelepon:
    def __init__(self):
        self.size = 8
        self.map = [None] * self.size

    def _probing(self, key):
        index = 1
        while index < self.size:
            probeHash = self._quadraticProbing(key, index)
            index += 1
            if (self.map[probeHash] is None) or (self.map[probeHash]=='none'):
                return probeHash
        index = 1
        while index < self.size:
            probeHash = self._linearProbing(key, index)
            index += 1
            if (self.map[probeHash] is None) or (self.map[probeHash]=='none'):
                return probeHash
        return None

    def _getHash(self, key):
        hash = 0
        for num in key:
            hash += int(num)
        return hash % self.size

    def _quadraticProbing(self, key, index):
        return (self._getHash(key)+index**2) % self.size

    def _linearProbing(self, key, index):
        return (self._getHash(key)+index) % self.size

    def add(self, key, value):
        key_hash = self._getHash(key)
        key_value = [key, value]
        if self.map[key_hash] is None:
            self.map[key_hash] = list(key_value)
        return True
```

```

    else:
        key_hash = self._probing(key)
        if key_hash is None:
            print("Slot sudah penuh!")
            return False
        self.map[key_hash] = list(key_value)
        return True

def getName(self, key):
    key_hash = self._getHash(key)
    if self.map[key_hash] is not None:
        if(self.map[key_hash][0] == key):
            return self.map[key_hash][1]
        else:
            for index in range(1,self.size+1):
                key_hash = self._quadraticProbing(key,index)
                if self.map[key_hash] is not None:
                    if(self.map[key_hash][0] == key):
                        return self.map[key_hash][1]

            for index in range(1,self.size+1):
                key_hash = self._linearProbing(key,index)
                if self.map[key_hash] is not None:
                    if(self.map[key_hash][0] == key):
                        return self.map[key_hash][1]
    print("Data Buku Telepon %s tidak ditemukan" %key)
    return 'Tidak ada'

def delete(self, key):
    key_hash = self._getHash(key)
    if self.map[key_hash] is not None:
        if(self.map[key_hash][0] == key):
            self.map[key_hash] = 'Sudah Dihapus'
            print("deleting ", key)
            return True
        else:
            for index in range(1,self.size+1):
                key_hash = self._quadraticProbing(key,index)
                if self.map[key_hash] is not None:
                    if(self.map[key_hash][0] == key):
                        self.map[key_hash] = 'none'
                        print("deleting ", key)
                        return True

            for index in range(1,self.size+1):

```

```

        key_hash = self._linearProbing(key, index)
        if self.map[key_hash] is not None:
            if (self.map[key_hash][0] == key):
                self.map[key_hash] = 'none'
                print("deleting ", key)
                return True
        print("Data Buku Telepon %s tidak ditemukan" %key)
        return(key, '= none')

def printAll(self):
    print('---Data Buku Telepon---')
    for item in self.map:
        if item is not None:
            print(str(item))
    print()

hashing = bukuTelepon()
hashing.add('081275663374', 'Dian')
hashing.add('082167665400', 'Dimas')
hashing.add('082167665420', 'Egi')
hashing.add('081363665130', 'Hennoch')
hashing.add('082164114312', 'Tiur')
hashing.add('082165225987', 'Ignes')
hashing.add('082165575660', 'Jebe')
hashing.add('089612430078', 'Ines')
hashing.add('089613738163', 'Caca')
hashing.printAll()
hashing.delete('081363665130')
hashing.delete('082165575660')
hashing.delete('082165575000')
hashing.printAll()
hashing.add('085765811220', 'Jessica')
hashing.printAll()
print('082165225987 = ' + hashing.getName('082165225987'))
print('082167665420 = ' + hashing.getName('082167665420'))
print('082167665000 = ' + hashing.getName('082167665000'))

```

```

PS C:\Users\M S I> & "C:/Users/M S I/AppData/Local/Programs/Python/Python39/python.exe" "d:/Semester 3/Praktikum Struktur Data/Laporan 7(1).py"
Slot sudah penuh!
---Data Buku Telepon---
['082165225987', 'Ignes']
['082164114312', 'Tiur']
['081363665130', 'Hennoch']
['082165575660', 'Jebe']
['081275663374', 'Dian']
['082167665400', 'Dimas']
['089612430078', 'Ines']
['082167665420', 'Egi']

deleting 081363665130
deleting 082165575660
Data Buku Telepon 082165575000 tidak ditemukan
---Data Buku Telepon---
['082165225987', 'Ignes']
['082164114312', 'Tiur']
Sudah Dihapus
Sudah Dihapus
['081275663374', 'Dian']
['082167665400', 'Dimas']
['089612430078', 'Ines']
['082167665420', 'Egi']

Slot sudah penuh!
---Data Buku Telepon---
['082165225987', 'Ignes']
['082164114312', 'Tiur']
Sudah Dihapus
Sudah Dihapus
['081275663374', 'Dian']
['082167665400', 'Dimas']
['089612430078', 'Ines']
['082167665420', 'Egi']

082165225987 = Ignés
082167665420 = Egi
Data Buku Telepon 082167665000 tidak ditemukan
082167665000 = Tidak ada
PS C:\Users\M S I> █

```

Penjelasan : Pada Soal Unguided 1 hash Table kita diminta untuk mengimplementasikan penanganan collision dengan menggunakan metode Quadratic Probe. pada function `_probing` akan valid apabila index adalah none atau ber-flag deleted. Setelah itu kita akan melakukan linear probing, fungsi `_linearProbing` disini akan dijalankan apabila lokasi yang kosong tidak ditemukan setelah kita menggunakan fungsi `_quadraticProbing`. Setelah itu kita akan menambahkan hash pada table. fungsi delete adalah untuk menghapus dengan melakukan linear probing. Setelah itu kita akan memeriksa apakah key adalah data yang akan dihapus. Pada data diatas kita memasukkan bukuTelepon. Didalam data tersebut kita telah menambahkan beberapa bukuTelepon beserta nama penggunanya. Setelah kita menghapus buku telepon, akan muncul pesan 'Sudah Dihapus' dari data tersebut. Untuk bukuTelepon yang tidak terdaftar akan mengeluarkan pesan Data buku Telepon tidak ditemukan ataupun tidak ada.

Daftar Pustaka :

http://aren.cs.ui.ac.id/sda/resources/sda2010/15_hashtable.pdf

Edwin Mahendra - 71200541