TUGAS MANDIRI

PERANCANGAN & ANALISIS ALGORITMA

“Open and Closed Hashing”

202323430048

DOSEN PENGAMPU:

Randi Proska Sandra, S.Pd, M.Sc

OLEH:

Ahmad Hanif Fajri

22343035

Informatika (NK)

PRODI SARJANA INFORMATIKA

DEPARTEMEN TEKNIK ELEKTRONIKA

FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

2024

A. PENJELASAN SINGKAT

Hopscotch Hashing adalah sebuah teknik dalam pengelolaan tabel hash yang memperkenalkan strategi baru untuk menangani masalah performa dan efisiensi dalam operasi penambahan, penghapusan, dan pencarian (contains). Dalam Hopscotch Hashing, setiap entri dalam tabel hash memiliki "bucket virtual" yang merupakan wilayah tetap dengan ukuran yang tumpang tindih dengan beberapa entri berikutnya. Konsep ini memungkinkan item yang di-hash ke suatu entri dapat ditemukan dengan cepat di dalam bucket virtual atau di salah satu bucket virtual berikutnya.

Algoritma Hopscotch Hashing menggunakan teknik pemindahan yang disebut "hopscotch". Saat menambahkan item baru, jika terdapat entri kosong dalam jarak tertentu dari entri saat ini, maka item tersebut ditempatkan di entri kosong tersebut. Namun, jika tidak ada entri kosong yang tersedia dalam jarak yang ditentukan, maka entri kosong yang paling dekat dengan entri saat ini akan dipindahkan lebih dekat ke bucket yang diinginkan. Strategi ini memungkinkan untuk menghindari masalah pengelompokan dan penurunan kinerja saat tabel mulai penuh, sebagaimana yang terjadi dalam algoritma linear probing.

Keunggulan dari Hopscotch Hashing meliputi:

1. Waktu akses yang konstan untuk operasi pencarian (contains), karena struktur tabel yang terorganisir dengan baik.

2. Kemudahan implementasi dan pemahaman, karena algoritma ini relatif sederhana dan mudah dimengerti.

3. Dukungan untuk beban yang lebih tinggi daripada beberapa algoritma hash lainnya, seperti cuckoo hashing, karena perpindahan item yang terjadi dalam hopscotch hashing tidak menghasilkan rantai perpindahan yang bersifat siklik.

B. PSEUDOCODE

ClosedHashTable

```
Class ClosedHashTable:
    Initialize(size):
        Set self.size = size
        Create an array self.table with size elements, initialized with None

    hash_function(key):
        Return key modulo self.size

    insert(key, value):
        index = hash_function(key)
```

```
        If self.table[index] is None:
            Set self.table[index] = (key, value)
        Else:
            While self.table[index] is not None:
                Increment index by 1 (index = (index + 1) modulo self.size)
            Set self.table[index] = (key, value)

    search(key):
        index = hash_function(key)
        original_index = index
        While self.table[index] is not None:
            k, v = self.table[index]
            If k equals key:
                Return v
            Increment index by 1 (index = (index + 1) modulo self.size)
            If index equals original_index:
                Break
        Return None
```

OpenHashTable

```
Class OpenHashTable:
    Initialize(size):
        Set self.size = size
        Create an array self.table with size elements, initialized with
None

    hash_function(key):
        Return key modulo self.size

    insert(key, value):
        index = hash_function(key)
        If self.table[index] is None:
            Create a new list [(key, value)] at self.table[index]
        Else:
            Append (key, value) to the list at self.table[index]

    search(key):
        index = hash_function(key)
        If self.table[index] is not None:
            For each (k, v) pair in the list at self.table[index]:
                If k equals key:
                    Return v
        Return None
```

C. SOURCE CODE

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

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

    def insert(self, key, value):
        index = self.hash_function(key)
        if self.table[index] is None:
            self.table[index] = [(key, value)]
        else:
            self.table[index].append((key, value))

    def search(self, key):
        index = self.hash_function(key)
        if self.table[index] is not None:
            for k, v in self.table[index]:
                if k == key:
                    return v
        return None

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

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

    def insert(self, key, value):
        index = self.hash_function(key)
        if self.table[index] is None:
            self.table[index] = (key, value)
        else:
            while self.table[index] is not None:
                index = (index + 1) % self.size
            self.table[index] = (key, value)
```

```
    def search(self, key):
        index = self.hash_function(key)
        original_index = index
        while self.table[index] is not None:
            k, v = self.table[index]
            if k == key:
                return v
            index = (index + 1) % self.size
            if index == original_index:
                break
        return None

# Contoh penggunaan OpenHashTable
open_table = OpenHashTable(10)
open_table.insert(5, "Nilai 5")
open_table.insert(15, "Nilai 15")
open_table.insert(25, "Nilai 25")

print(open_table.search(5))  # Output: Nilai 5
print(open_table.search(15)) # Output: Nilai 15
print(open_table.search(25)) # Output: Nilai 25

# Contoh penggunaan ClosedHashTable
closed_table = ClosedHashTable(10)
closed_table.insert(5, "Nilai 5")
closed_table.insert(15, "Nilai 15")
closed_table.insert(25, "Nilai 25")

print(closed_table.search(5))  # Output: Nilai 5
print(closed_table.search(15)) # Output: Nilai 15
print(closed_table.search(25)) # Output: Nilai 25
```

Hasil Screenshot :

```
PS C:\Users\user> & C:/Users/user/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/user/Documents/Smstr 4 Fajri/Pak Randy/OpenHashTable dan ClosedHashTab
le.py"
Nilai 5
Nilai 15
Nilai 25
Nilai 5
Nilai 15
Nilai 25
PS C:\Users\user>
```

D. ANALISIS KEBUTUHAN WAKTU

1. OpenHashTable:

- Inisialisasi:

- Kebutuhan: Pengguna harus memberikan ukuran (size) saat membuat objek OpenHashTable.

- Fungsi: Menciptakan tabel hash dengan ukuran yang ditentukan dan menginisialisasi setiap elemen dengan nilai None.

- hash_function:

- Kebutuhan: Fungsi ini memerlukan input berupa kunci (key).

- Fungsi: Mengembalikan indeks hash yang dihitung berdasarkan modulus ukuran tabel.

- insert:

- Kebutuhan: Memerlukan kunci (key) dan nilai (value) yang ingin dimasukkan.

- Fungsi: Memasukkan pasangan kunci-nilai ke dalam tabel hash menggunakan metode chaining jika terjadi collision.

- search:

- Kebutuhan: Memerlukan kunci (key) yang ingin dicari.

- Fungsi: Mencari nilai yang sesuai dengan kunci yang diberikan dalam tabel hash menggunakan algoritma pencarian linear pada rantai (chaining).

2. ClosedHashTable:

-Inisialisasi:

- Kebutuhan: Pengguna harus memberikan ukuran (size) saat membuat objek ClosedHashTable.

- Fungsi: Menciptakan tabel hash dengan ukuran yang ditentukan dan menginisialisasi setiap elemen dengan nilai None.

-hash_function:

- Kebutuhan: Fungsi ini memerlukan input berupa kunci (key).

- Fungsi: Mengembalikan indeks hash yang dihitung berdasarkan modulus ukuran tabel.

- insert:

- Kebutuhan: Memerlukan kunci (key) dan nilai (value) yang ingin dimasukkan.

- Fungsi: Memasukkan pasangan kunci-nilai ke dalam tabel hash menggunakan metode linear probing jika terjadi collision.

- search:

- Kebutuhan: Memerlukan kunci (key) yang ingin dicari.

- Fungsi: Mencari nilai yang sesuai dengan kunci yang diberikan dalam tabel hash menggunakan algoritma linear probing.

E. REFERENSI

HERLIHY, Maurice; SHAVIT, Nir; TZAFRIR, Moran. Hopscotch hashing. In: *Distributed Computing: 22nd International Symposium, DISC 2008, Arcachon, France, September 22-24, 2008. Proceedings 22*. Springer Berlin Heidelberg, 2008. p. 350-364.

F. LINK GITHUB

https://github.com/Jriii19/Open-and-Closed-Hashing