14.1What is Hashing?

Là một cấu trúc dữ liệu tương tự như mảng nhưng được lập chỉ mục theo các dạng dữ liệu khác như chuỗi thay vì số nguyên. Bảng băm là một cấu trúc dữ liệu phổ biến để triển khai mảng kết hợp. Để phân tích hiệu quả tiệm cận của bảng băm, chúng ta cần xem xét độ phức tạp trường hợp trung bình. Băm là một kỹ thuật được sử dụng để lưu trữ và truy xuất thông tin nhanh nhất có thể, hữu ích trong việc thực hiện tìm kiếm tối ưu và triển khai các bảng ký hiệu.

14.2 Why Hashing?

Trong Python, chúng ta đã thấy rằng cây tìm kiếm nhị phân cân bằng hỗ trợ các hoạt động như chèn, xóa và tìm kiếm trong thời gian O(logn). Trong nhiều ứng dụng, nếu chúng ta cần thực hiện những thao tác này trong O(1), thì hàm băm có thể là một giải pháp. Hãy nhớ rằng độ phức tạp của hàm băm trong trường hợp xấu nhất vẫn là O(n), nhưng trung bình nó cho kết quả là O(1).

14.3 Hash Table ADT

Cấu trúc bảng băm là một tập hợp không có thứ tự các liên kết giữa khóa và giá trị dữ liệu. Các khóa trong bảng băm đều là duy nhất nên có mối quan hệ một-một giữa khóa và giá trị. Các hoạt động được đưa ra dưới đây.

1. HashTable: Tạo bảng băm mới
2. Get: Tìm kiếm bảng băm bằng khóa và trả về giá trị nếu tìm thấy phần tử có khóa đã cho
3. Put: Chèn một cặp khóa-giá trị mới vào bảng băm
4. Delete: Xóa cặp khóa-giá trị khỏi bảng băm
5. DeleteHashTable: Xóa bảng băm

Cú pháp ví dụ:

class HashTable:

    def \_\_init\_\_(self):

        self.table = {}

    def get(self, key):

        return self.table.get(key, None)

    def put(self, key, value):

        self.table[key] = value

    def delete(self, key):

        if key in self.table:

            del self.table[key]

    def deleteHashTable(self):

        self.table = {}

# Sử dụng:

# Tạo bảng băm mới

hash\_table = HashTable()

# Chèn cặp khóa-giá trị

hash\_table.put("key1", "value1")

# Tìm kiếm và in giá trị

result = hash\_table.get("key1")

print("Giá trị của key1:", result)

# Xóa cặp khóa-giá trị

hash\_table.delete("key1")

# Kiểm tra xem key1 có còn trong bảng băm không

result\_after\_delete = hash\_table.get("key1")

print("Giá trị của key1 sau khi xóa:", result\_after\_delete)

# Xóa bảng băm

hash\_table.deleteHashTable()

14.4 Understanding Hashing

Mô tả một thuật toán sử dụng mảng như một bảng băm để tìm ký tự lặp lại đầu tiên trong một chuỗi. Độ phức tạp thời gian của thuật toán là O(n²) và độ phức tạp không gian là O(1). Thuật toán tạo một mảng có kích thước 256 (tương ứng với số lượng ký tự ASCII), khởi tạo tất cả các giá trị là 0. Khi quét qua chuỗi đầu vào, thuật toán tăng số lượng của ký tự tương ứng trong mảng. Thời gian để truy cập bất kỳ vị trí nào trong mảng là không đổi. Ký tự lặp lại đầu tiên sẽ là ký tự có số lượng lớn hơn 1 sau khi quét hết chuỗi đầu vào.

def first\_repeated\_char(input\_str):

    count = [0] \* 256  # Additional array

    for char in input\_str:

        count[ord(char)] += 1

    for char in input\_str:

        if count[ord(char)] == 1:

            print(char)

            return char

    print("No Repeated Characters")

    return 0

# Example usage:

input\_string = "abcdefgah"

result = first\_repeated\_char(input\_string)

if result:

    print("First Repeated Character:", result)

14.4 Components of Hashing

Băm là một quy trình bao gồm bốn thành phần chính:

1. **Bảng Băm**: Đây là cấu trúc dữ liệu lưu trữ các cặp khóa-giá trị. Bảng băm sử dụng hàm băm để tính toán chỉ số, từ đó giá trị mong muốn có thể được tìm thấy.
2. **Hàm Băm**: Hàm băm nhận một đầu vào (hoặc “khóa”) và tạo ra một chuỗi kí tự có kích thước cố định, thường là một mã băm. Mục tiêu là phân phối các khóa đồng đều trên bảng băm.
3. **Va Chạm**: Va chạm xảy ra khi hai khóa khác nhau băm đến cùng một chỉ số trong bảng băm. Va chạm có thể dẫn đến không hiệu quả và cần được xử lý một cách phù hợp.
4. **Các Kỹ Thuật Giải Quyết Va Chạm**: Những kỹ thuật này được sử dụng để giải quyết va chạm và đảm bảo mỗi khóa được lưu trữ ở một vị trí duy nhất trong bảng băm. Một số kỹ thuật phổ biến bao gồm “chaining” (sử dụng danh sách liên kết để xử lý nhiều khóa tại cùng một chỉ số) và “open addressing” (kiểm tra và tìm kiếm vị trí trống tiếp theo).