Building a HashMap in C

using AVL trees, linked lists, and the djb2 hashing algorithm

Data Structures used

AVL Tree (primary)

Linked Lists (secondary)

We use an AVL tree to store the hashed data, and Linked Lists to handle hash collisions through open-chaining. We use AVL tree instead of normal binary search tree because it is self-balancing.

API (Application Programming Interface)

Insert

Input: object and it’s key
Complexity: O(log n)

Remove

Input: object and it’s key
Complexity: O(log n)

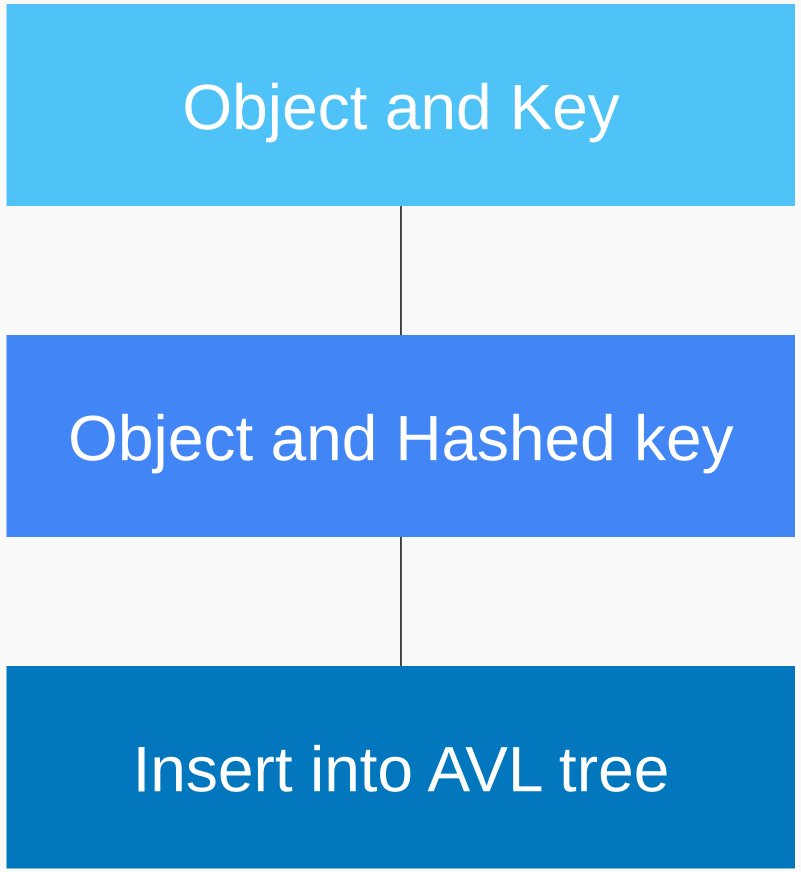
Search

Input: object and it’s key
Complexity: O(log n)

Input/Output for Insert

```
void insert(void *object, void *key,
size_t len);
```

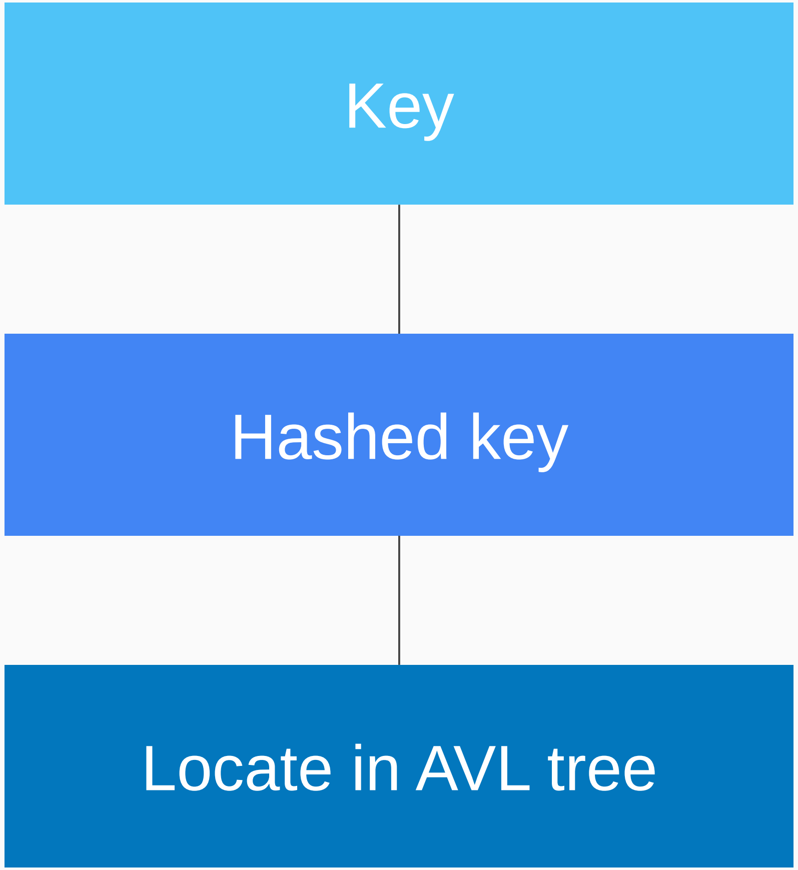
Input/Output for Insert



Input/Output for Search

```
void* search(void *key, size_t len);
```

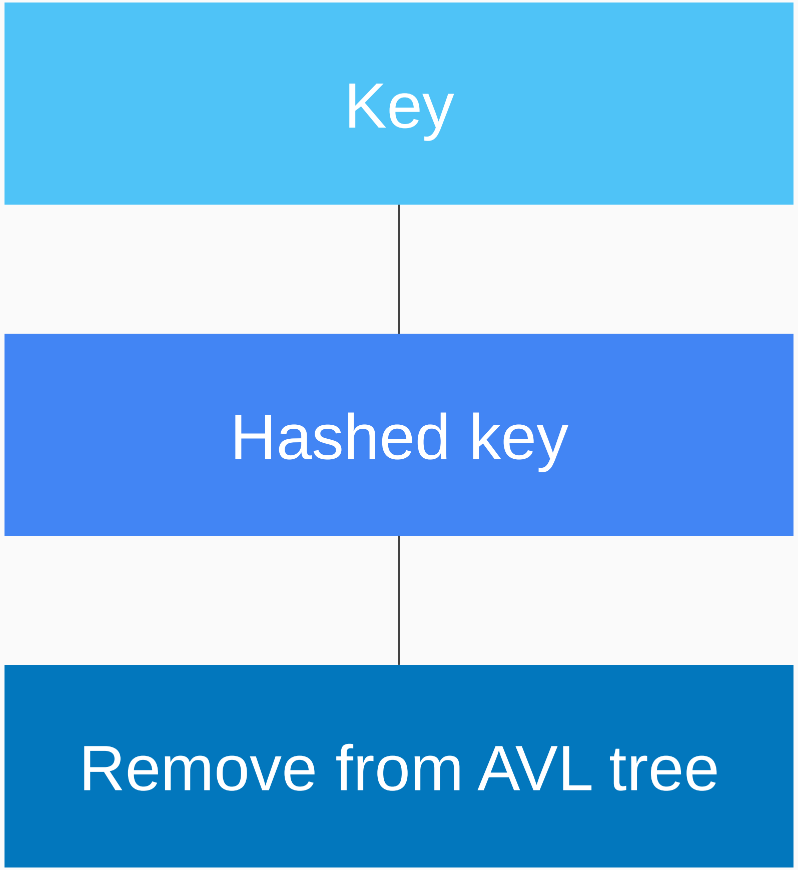
Input/Output for Search



Input/Output for Remove

```
void remove(void *key, size_t len);
```

Input/Output for Remove



Node for AVL Tree

```
struct node {
void *data; // object
unsigned long hkey; // hashed key
struct node *right;
struct node *left;
}
```

The djb2 hashing algorithm

an efficient hashing algorithm disc
overed by Dan Bernstein

Input: a pointer to an array of bytes

(unsigned char *)

Output: a unique unsigned integer

(unsigned long)

The djb2 hashing algorithm

an efficient hashing algorithm disc
overed by Dan Bernstein

```
unsigned long
hash(unsigned char *str, size_t len) {
    unsigned long hash = 5381;
    int i = 0;

    while (i < len) {
        hash = ((hash << 5) + hash) + c;
                // hash * 33 + c
        i++;
        str++;
    }
    return hash;
}
```