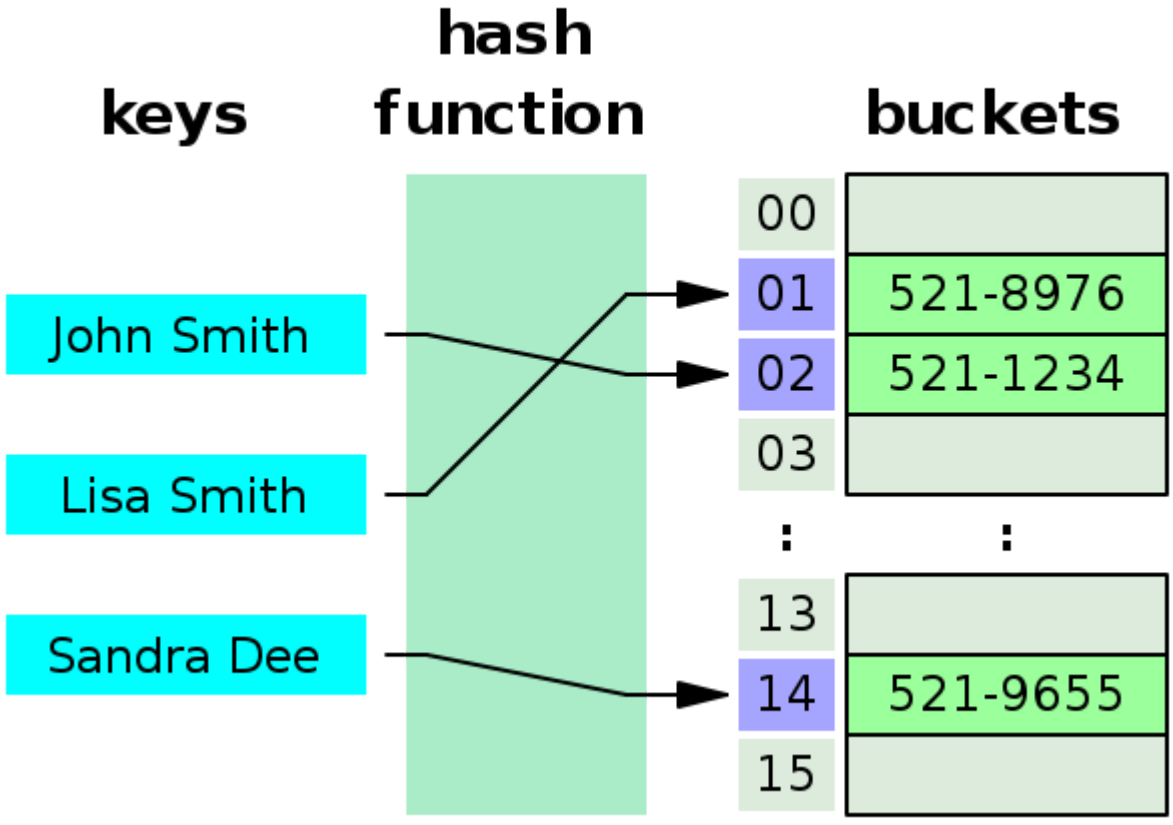
Bảng băm (Hash Table)

Bảng băm (Hash Table)

Tư tưởng

Bảng băm là một CTDL thường được sử dụng như một từ điển: mỗi phần tử trong bảng băm là một cặp (khóa, giá trị). Nếu so sánh với mảng, khóa được xem như chỉ số của mảng, còn giá trị giống như giá trị mà ta lưu tại chỉ số tương ứng. Bảng băm không như các loại từ điển thông thường - ta có thể tìm được giá trị thông qua khóa của nó.

Không may, không phải tất cả các kiểu dữ liệu đều có thể sắp xếp vào một từ điển đơn giản. Đây chính là lúc mà quá trình băm (hash) ra đời. Hash là quá trình khởi tạo một giá trị khóa (thường là 32 bit hoặc 64 bit) từ một phần dữ liệu. Nó có thể là n bit đầu tiên của dữ liệu, n bit cuối cùng, giá trị mod cho một số nguyên tố nào đó. Dựa theo giá trị hash, dữ liệu được chia vào các **bucket**:



Giải thích hình minh họa:

- Ta cần lưu số điện thoại của 3 người:

    - John Smith: 521-1234
    - Lisa Smith: 521-8976
    - Sandra Dee: 521-9655
- Giá trị Hash của 3 người này lần lượt là: 1, 2 và 14.
- Sau khi tính được giá trị Hash của 3 người, ta lưu vào các bucket tương ứng là 1, 2 và 14.

Nếu các kết quả của hàm hash được phân bố đều, các bucket sẽ có kích thước xấp xỉ nhau. Giả sử số bucket đủ lớn, mỗi buckets sẽ chỉ có một vài phần tử trong chúng. Điều này làm cho việc tìm kiếm rất hiệu quả.

Độ phức tạp

Gọi:

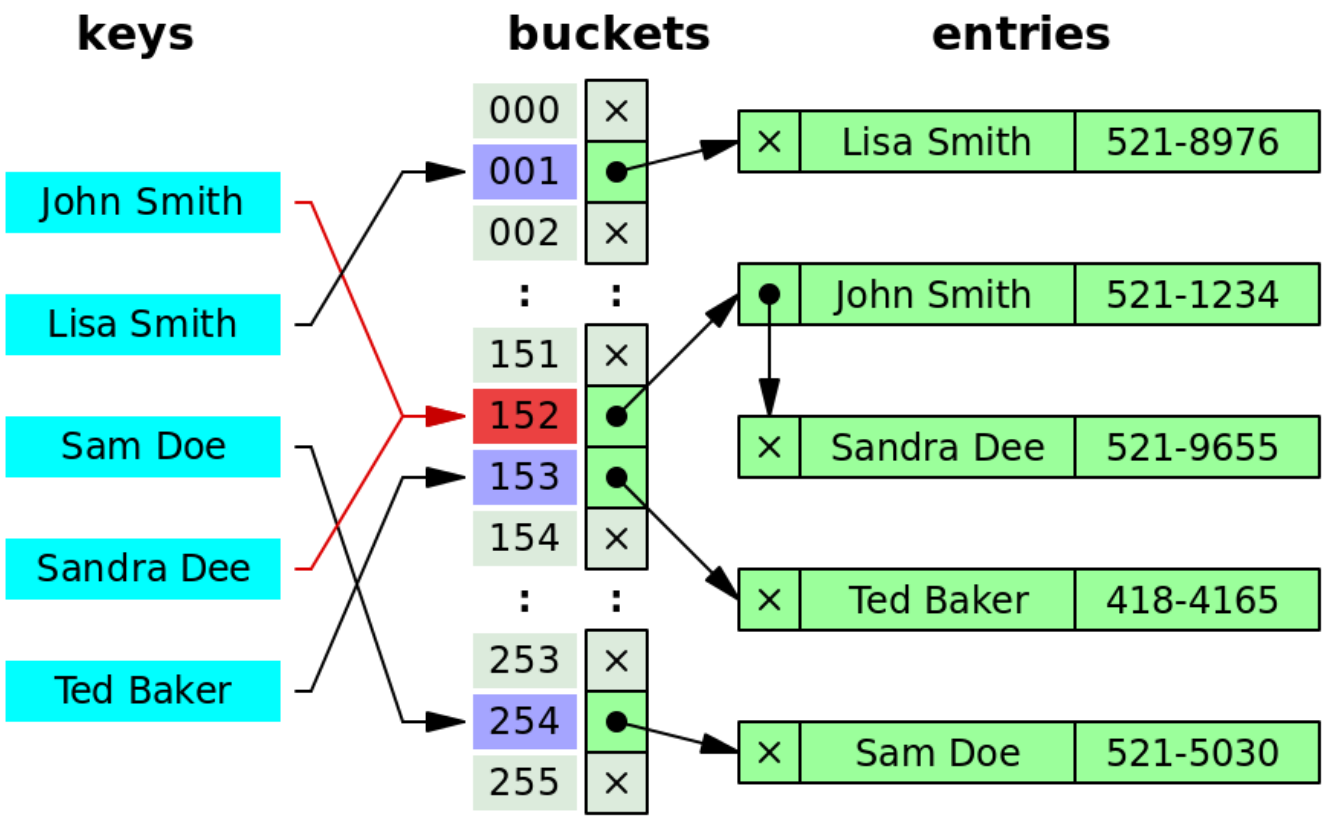
- n là số phần tử ta cần lưu trong Hash table
- k là số bucket

Giá trị n/k được gọi là **load factor**. Khi load factor nhỏ (xấp xỉ 1), và giá trị của hàm Hash phân bố đều, độ phức tạp của các thao tác trên Hash table là $\mathcal{O}(1)$.

Hash collision

Separate chaining

Trường hợp một hash bucket chứa nhiều hơn một giá trị ta gọi đó là **Hash collision** (va chạm). Việc xử lý hash collision rất quan trọng đối với độ hiệu quả của bảng băm. Một trong những phương pháp đơn giản nhất là cài đặt các danh sách liên kết ở các bucket. Kĩ thuật này được gọi là **Separate chaining**:

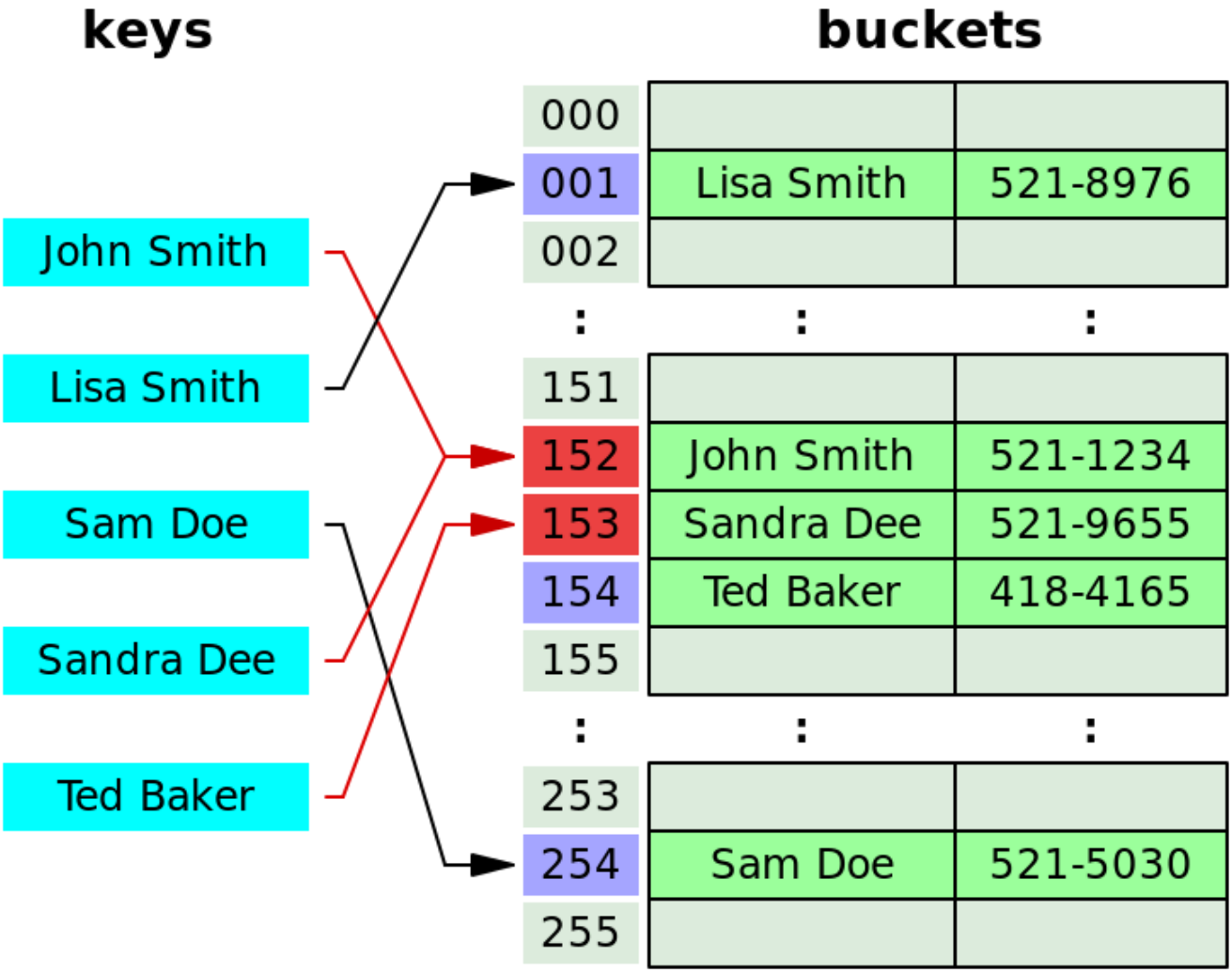


Giải thích hình minh họa:

- Mỗi bucket là 1 danh sách liên kết
- John Smith và Sandra Dee cùng có giá trị hàm hash là 152, nên ở bucket 152, ta có 1 danh sách liên kết chứa 2 phần tử.

Open Addressing

Tư tưởng của **Open Addressing** là, khi xảy ra Hash collision, ta lưu vào một vị trí tiếp theo trong bảng băm. Ví dụ:



Trong hình minh họa:

- John Smith và Sandra Dee đều có giá trị Hash là 152. Vì ta đã lưu John Smith vào bucket 152, nên ta buộc phải lưu Sandra Dee vào bucket 153.
- Ted Baker có giá trị Hash là 153, nhưng lúc này bucket 153 đã lưu thông tin của Sandra Dee, nên ta phải lưu giá trị của Ted Baker vào bucket 154.

Chú ý:

- Để cài đặt được cách này, Load factor phải nhỏ hơn 1.
- Khi tìm kiếm 1 phần tử, ta phải kiểm tra tất cả các bucket bắt đầu từ bucket của giá trị Hash, đến khi bucket trống (ví dụ ta tìm Sandra Dee thì phải tìm bucket 152, 153; tìm Ted Baker thì phải tìm bucket 152, 153, 154. Nếu ta tìm 1 người khác có giá trị Hash là 152, thì phải tìm cả bucket 152, 153, 154 và 155 (chỉ khi bucket 155 trống, ta mới chắc chắn người đó không có trong Hash table).

Cài đặt

Dưới đây là cài đặt Hash table đơn giản, hỗ trợ thao tác thêm và tìm kiếm. Hash table này sử dụng separate chaining, và dùng vector thay cho linked list để đơn giản.

```cpp
const int P = 1e6 + 3;

struct HashTable {
    vector< pair<int,int> > h[P];

public:
    void insert(int key, int value) {
        int hkey = getHash(key);
        for (auto p : h[hkey]) {
            if (p.first == key) {
                // key da ton tai trong Hash table, ta bo qua
                return;
            }
        }
        // Them (key, value) vao hash table
        h[hkey].emplace_back(key, value);
    }

    int find(int key) {
        int hkey = getHash(key);
        for(auto p : h[hkey]) {
            if (p.first == key) {
                // ton tai key trong Hash table, return value
                return p.value;
            }
        }
        // Khong tim thay
        return 0;
    }

private:
    int getHash(int key) {
        // Cho 1 key, tra lai Hash value la key % P
        return key % P;
    }
};
```

Kết luận

Trong phát triển ứng dụng, bảng hash thuận tiện để lưu trữ dữ liệu tham khảo, chẳng hạn như chữ viết tắt tên đầy đủ của các tổ chức. Trong giải quyết bài toán, bảng hash thật sự hữu ích để tiếp cận hướng giải quyết chia để trị cho bài toán cái túi (knapsack-type).

Giả sử, chúng ta được yêu cầu tìm một số lượng ống nhỏ nhất để xây dựng một đường ống với chiều dài cố định và chúng ta có 38 ống. Bằng cách chia thành hai tập – 19 và tính toán tất cả trường hợp ghép ống có thể ở mỗi tập, chúng ta tạo ra một bảng hash kết nối giữa độ dài mà các tổ hợp ống tạo thành với số lượng ống ít nhất cần dùng. Sau đó, với mỗi tổ hợp ống được xây dựng trong một tập, chúng ta có thể tra cứu liệu có tồn tại đường ống với độ dài phù hợp ở tập bên kia để cả hai ống tạo nên một đường ống với độ dài yêu cầu với số ống là ít nhất.

Các tài liệu tham khảo:

- Topcoder
- Wikipedia