



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

Tìm kiếm

Nội dung

- Tìm kiếm tuần tự
- Tìm kiếm nhị phân
- Cây nhị phân tìm kiếm
- Bảng băm

Tìm kiếm tuần tự

- Cho dãy $X[L..R]$ và một giá trị Y .
Tìm chỉ số i sao cho $X[i] = Y$

```
sequentialSearch(X[], int L, int R,  
    int Y) {  
    for(i = L; i <= R; i++)  
        if(X[i] = Y) return i;  
    return -1;  
}
```

Tìm kiếm nhị phân

- Dãy đối tượng được sắp xếp theo thứ tự không tang (hoặc không giảm) của giá trị khóa
- Dựa trên chia để trị:
 - So sánh khóa đầu vào Y với phần tử ở chính giữa của dãy, và quyết định tiếp tục tìm kiếm ở nửa bên trái hoặc nửa bên phải đối với phần tử ở chính giữa
- Độ phức tạp thời gian: $O(\log n)$

```
binarySearch(X[], int L, int R,  
             int Y) {  
    if(L = R){  
        if(X[L] = Y) return L;  
        return -1;  
    }  
    int mid = (L+R)/2;  
    if(X[mid] = Y) return mid;  
    if(X[mid] < Y)  
        return binarySearch(X,mid+1,R,Y);  
    return binarySearch(X,L,mid-1,Y);  
}
```

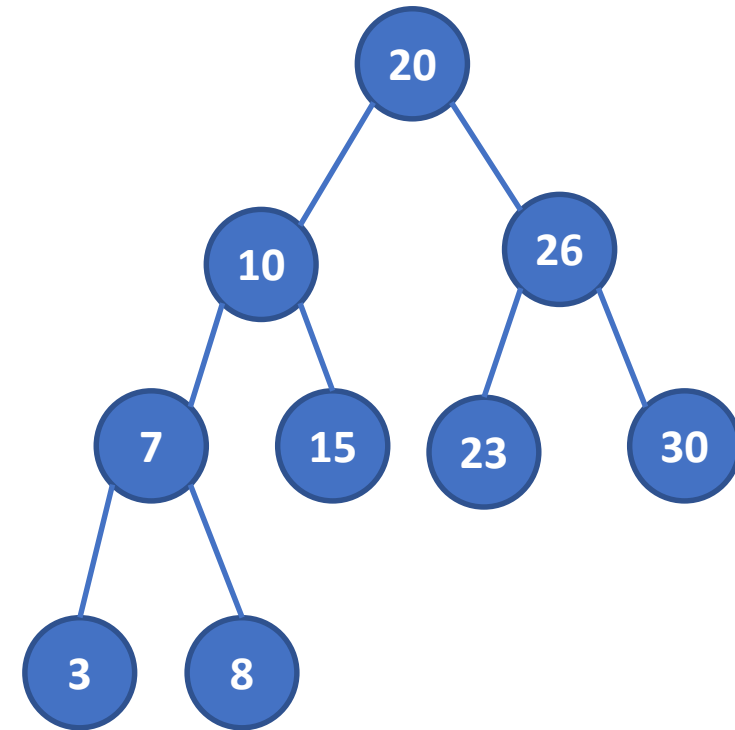
Tìm kiếm nhị phân

- **Bài tập** Cho dãy số gồm các phần tử đôi một khác nhau a_1, a_2, \dots, a_N và một giá trị b . Đếm số cặp (a_i, a_j) sao cho $a_i + a_j = b$ ($i < j$)

Cây nhị phân tìm kiếm - BST

- BST là một cấu trúc dữ liệu lưu trữ các đối tượng dưới dạng cây nhị phân:
 - Khóa của mỗi nút lớn hơn khóa của tất cả các nút ở cây con trái và nhỏ hơn khóa của tất cả các nút ở cây con bên phải

```
struct Node{  
    int key;  
    Node* leftChild;  
    Node* rightChild;  
};  
Node* root;
```



Cây nhị phân tìm kiếm - BST

- Các thao tác
 - Node* makeNode(int v): tạo ra một nút mới có khóa v
 - Node* insert(Node* r, int v): tạo ra 1 nút có khóa là v và chèn vào BST có gốc là r
 - Node* search(Node* r, int v): tìm và trả về nút có khóa bằng v trong BST gốc r
 - Node* findMin(Node* r): tìm và trả về nút có khóa nhỏ nhất trên BST gốc r
 - Node* del(Node* r, int v): loại bỏ nút có khóa bằng v khỏi BST gốc r

Cây nhị phân tìm kiếm - BST

```
Node* makeNode(int v) {  
    Node* p = new Node;  
    p->key = v;  
    p->leftChild = NULL;  
    p->rightChild = NULL;  
    return p;  
}
```

```
Node* insert(Node* r, int v) {  
    if(r == NULL)  
        r = makeNode(v);  
    else if(r->key > v)  
        r->leftChild = insert(r->leftChild,v);  
    else if(r->key <= v)  
        r->rightChild = insert(r->rightChild,v);  
    return r;  
}
```


Cây nhị phân tìm kiếm - BST

```
Node* search(Node* r, int v) {  
    if(r == NULL)  
        return NULL;  
    if(r->key == v)  
        return r;  
    else if(r->key > v)  
        return search(r->leftChild, v);  
    return search(r->rightChild, v);  
}
```

```
Node* findMin(Node* r) {  
    if(r == NULL)  
        return NULL;  
    Node* lmin = findMin(r->leftChild);  
    if(lmin != NULL) return lmin;  
    return r;  
}
```

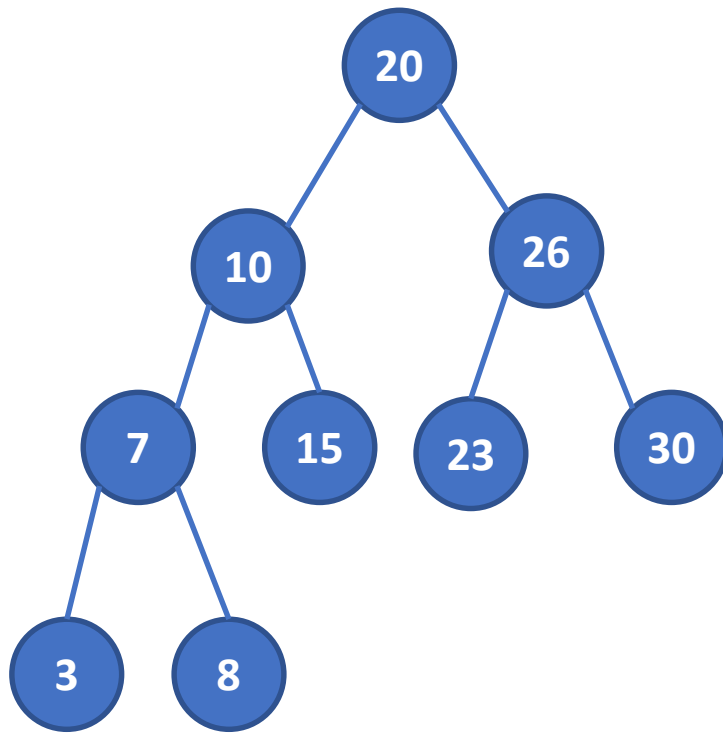
Cây nhị phân tìm kiếm - BST

```
Node* del(Node* r, int v) {
    if(r == NULL) return NULL;
    else if(v < r->key) r->leftChild = del(r->leftChild, v);
    else if(v > r->key) r->rightChild = del(r->rightChild, v);
    else{
        if(r->leftChild != NULL && r->rightChild != NULL){
            Node* tmp = findMin(r->rightChild);
            r->key = tmp->key; r->rightChild = del(r->rightChild, tmp->key);
        }else{
            Node* tmp = r;
            if(r->leftChild == NULL) r = r->rightChild;
            else r = r->leftChild;
            delete tmp;
        }
    }
    return r;
}
```

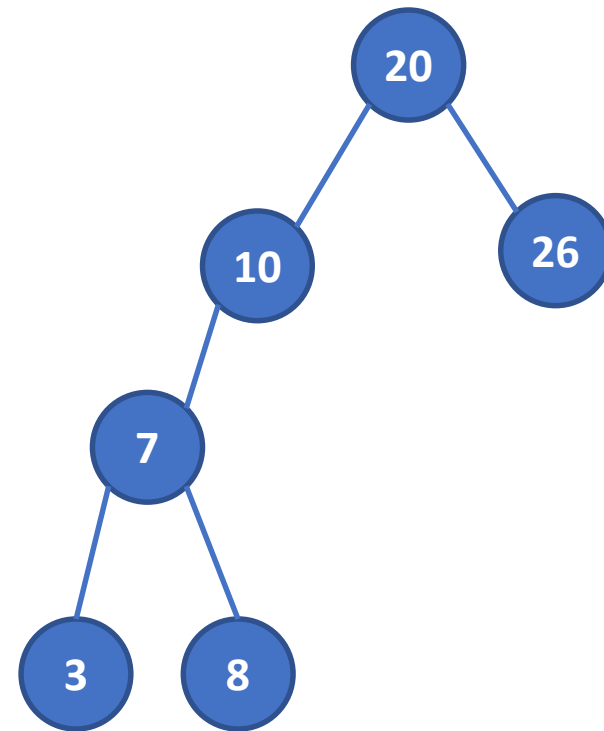
Cây nhị phân tìm kiếm cân bằng - AVL

- AVL là một BST với thuộc tính cân bằng
 - Chênh lệch khóa của nút con trái và con phải của mỗi nút nhiều nhất là 1 đơn vị
 - Chiều cao của AVL là $\log N$ (N là số nút của cây)
 - Thao tác thêm 1 phần tử hoặc loại bỏ 1 phần tử khỏi AVL phải bảo tồn thuộc tính cân bằng

Cây nhị phân tìm kiếm cân bằng - AVL



AVL



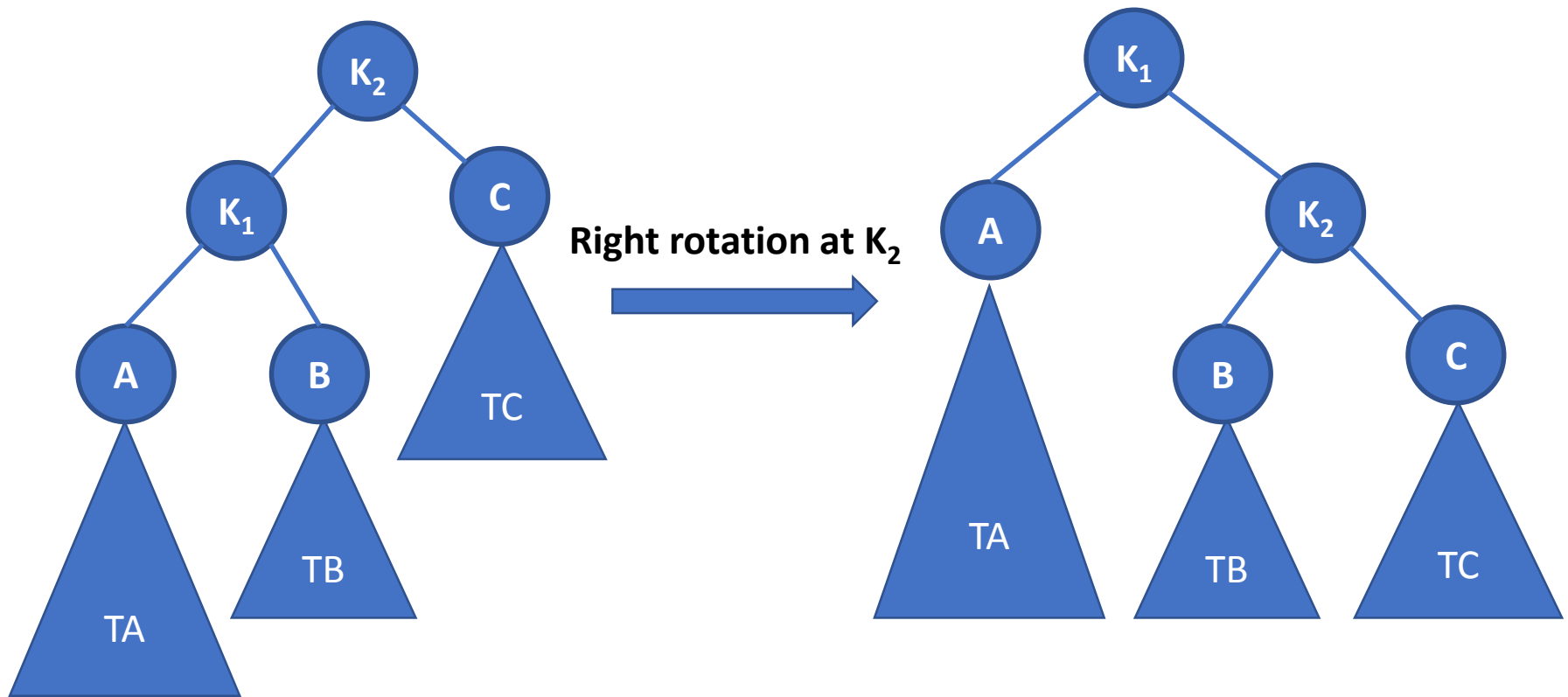
BST nhưng không phải AVL

Cây nhị phân tìm kiếm cân bằng - AVL

- Chênh lệch độ cao của 2 nút con của một nút nào đó có thể bằng 2 sau khi thêm mới hoặc loại bỏ 1 nút khỏi AVL
- Thực hiện các phép xoay để khôi phục thuộc tính cân bằng của AVL

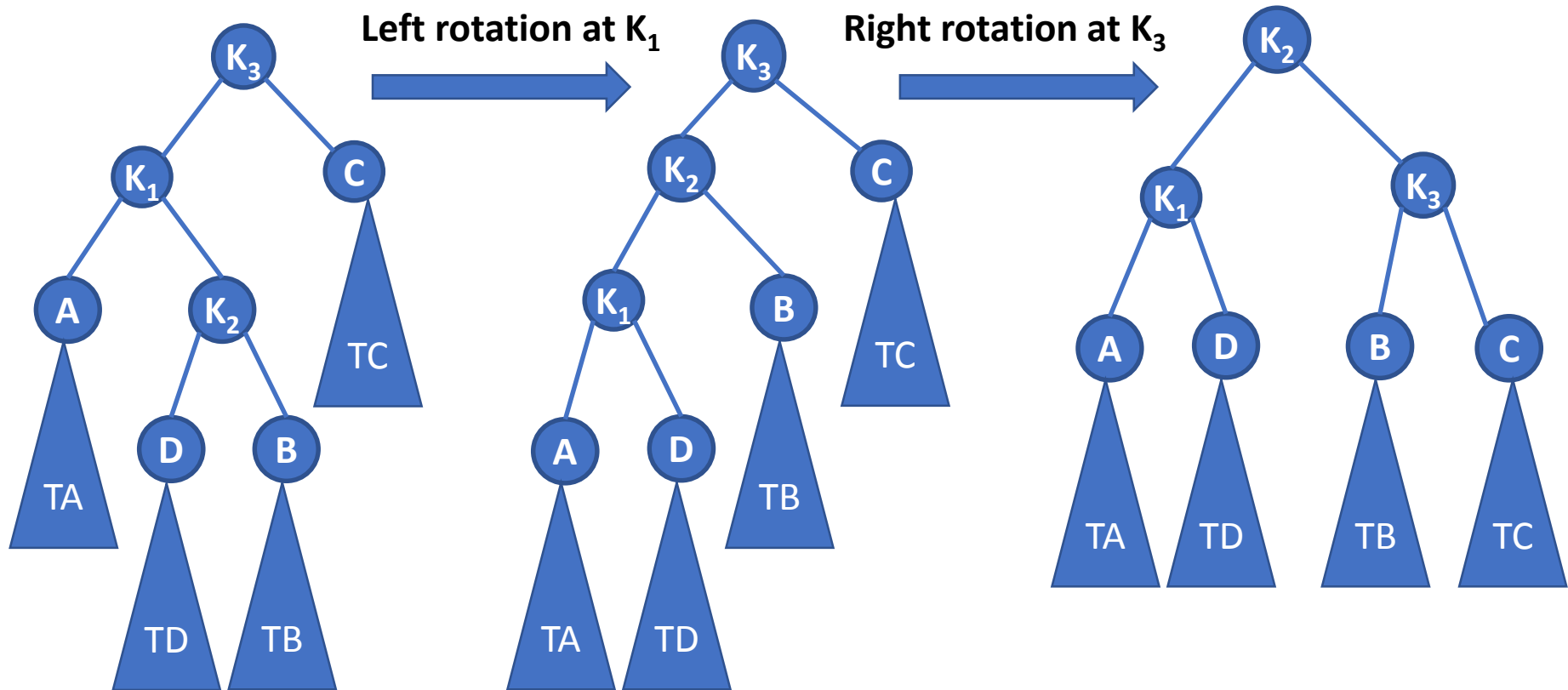
Cây nhị phân tìm kiếm cân bằng - AVL

Case 1: Thực hiện xoay phải



Balanced Binary Search Tree - AVL

Case 2

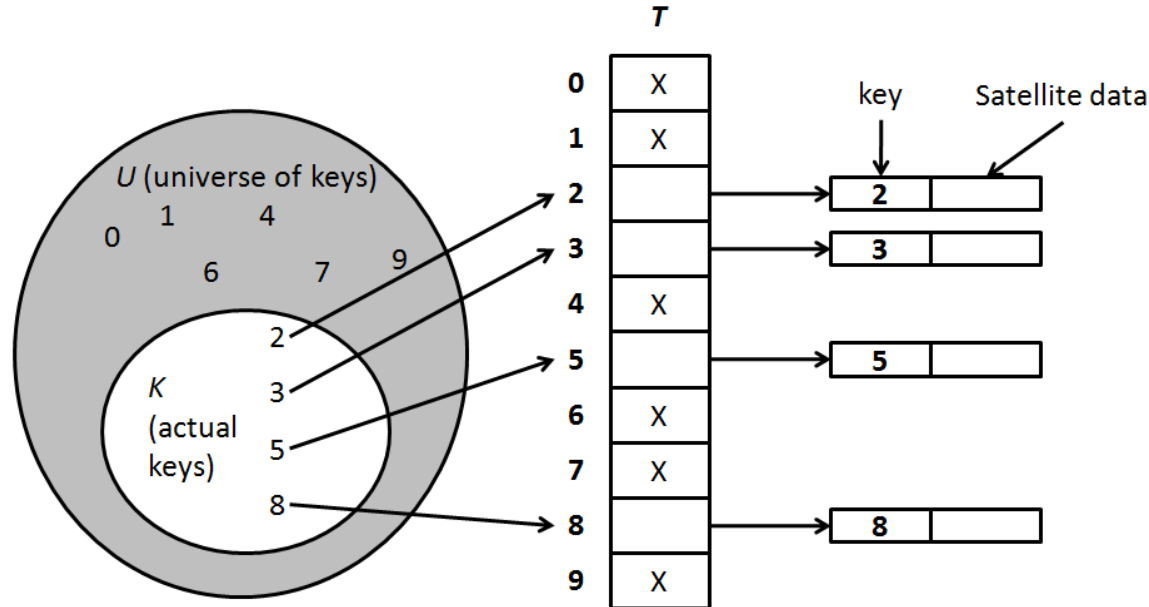


Bảng băm

- Ánh xạ: một cấu trúc dữ liệu lưu trữ các cặp khóa – giá trị (key, value)
 - $\text{put}(k, v)$: Ánh xạ khóa k với giá trị v
 - $\text{get}(k)$: trả về giá trị tương ứng với khóa k
- Cài đặt
 - Cây nhị phân tìm kiếm
 - **Bảng băm**

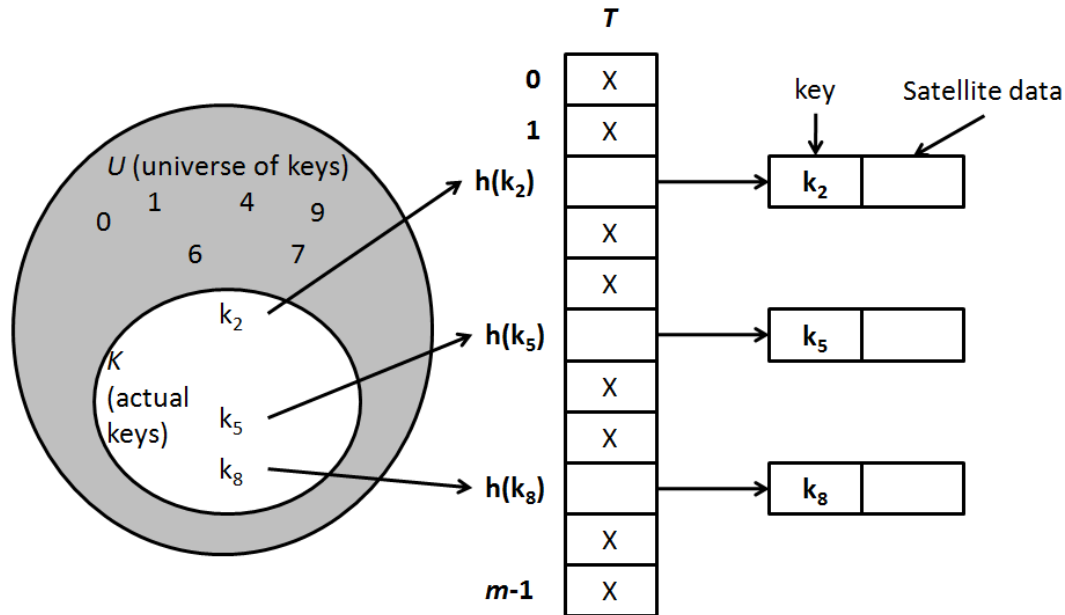
Bảng băm

- Địa chỉ trực tiếp
 - Giá trị của khóa k được sử dụng làm địa chỉ trực tiếp, xác định vị trí cặp khóa – giá trị (k,v) được lưu trữ
 - Ưu điểm: đơn giản, tìm kiếm nhanh
 - Nhược điểm: hiệu quả sử dụng bộ nhớ kém khi miền giá trị của khóa trải rộng và số lượng khóa được dung rất ít



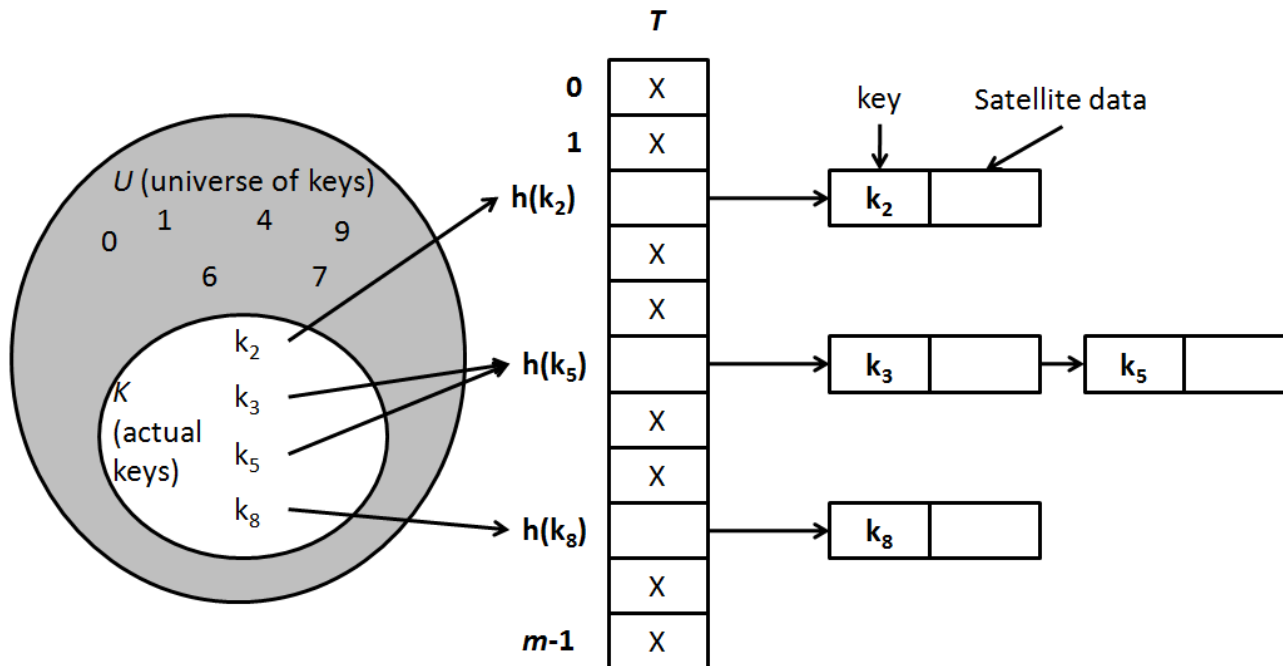
Bảng băm

- Hàm băm $h(k)$ xác định địa chỉ nơi $(k, value)$ được lưu trữ
- $h(k)$ cần đơn giản và tính toán hiệu quả



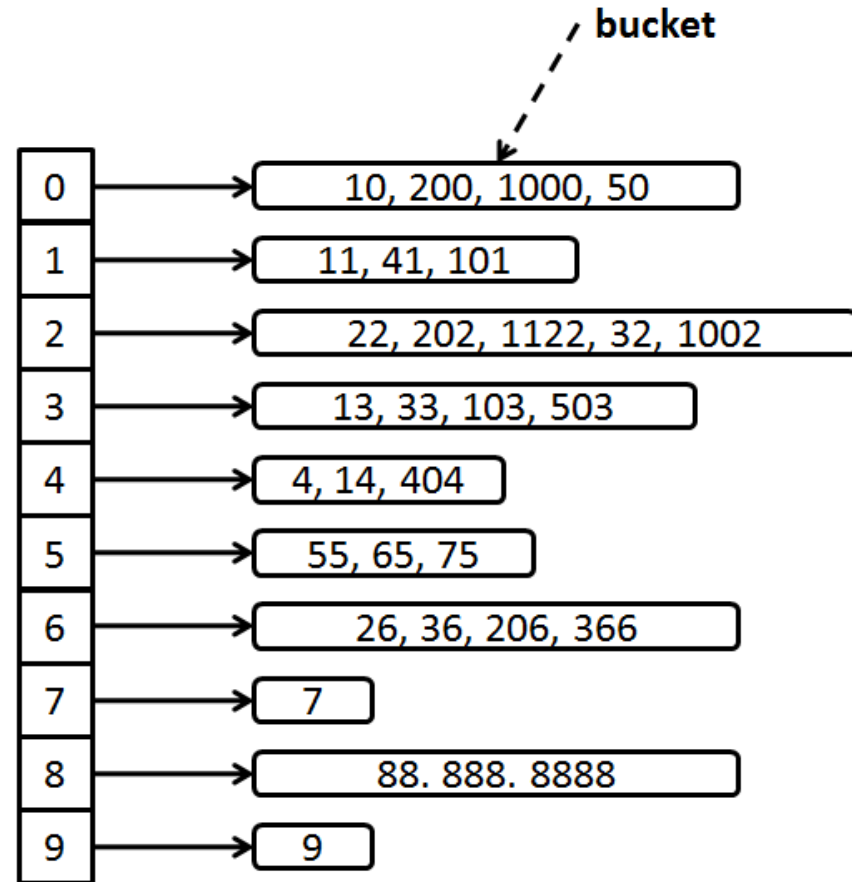
Bảng băm

- Xung đột: hai khóa khác nhau cho cùng giá trị hàm băm
- Giải quyết xung đột:
 - Nhóm chuỗi (chaining): nhóm các khóa có cùng giá trị hàm băm vào các cụm
 - Địa chỉ mở (Open addressing)



Hashing

- Modulo: $h(k) = k \bmod m$ trong đó m là kích thước bảng lưu trữ



Bảng băm: địa chỉ mở

- Cặp (key, value) được lưu vào các slot của bảng
- Các thao tác $put(k, v)$ và $get(k)$ cần thực hiện việc dò (probe) để tìm ra vị trí mong muốn trong bảng nơi lưu trữ khóa – giá trị
 - $put(k, v)$: dò để tìm ra vị trí trống để lưu (k, v)
 - $get(k)$: dò để tìm ra vị trí trong bảng nơi k được lưu trữ
 - Thứ tự dò: $h(k, 0), h(k, 1), h(k, 2), \dots, h(k, m-1)$
 - Các phương pháp dò
 - Dò tuyến tính: $h(k, i) = (h_1(k) + i) \bmod m$ trong đó h_1 làm hàm băm thông thường
 - Dò toàn phương: $h(k, i) = (h_1(k) + c_1i + c_2i^2) \bmod m$ trong đó h_1 là hàm băm thông thường
 - Băm kép: $h(k, i) = (h_1(k) + i * h_2(k)) \bmod m$ trong đó h_1 và h_2 là các hàm băm thông thường

Bảng băm: địa chỉ mở

```
get(k)
{
    // T: the table
    i = 0;
    while(i < m) {
        j = h(k,i);
        if(T[j].key = k) {
            return T[j];
        }
        i = i + 1;
    }
    return NULL;
}
```

```
put(k, v)
{
    // T: the table
    x.key = k; x.value = v;
    i = 0;
    while(i < m) {
        j = h(k,i);
        if(T[j] = NULL) {
            T[j] = x; return j;
        }
        i = i + 1;
    }
    error("Hash table overflow");
}
```

Bảng băm: địa chỉ mở

- Bài tập: Một bảng có m ô lưu trữ, áp dụng chiến lược địa chỉ mở với $h(k, i)$ có dạng:

$$h(k, i) = (k \bmod m + i) \bmod m$$

- Ban đầu, bảng trống rỗng, hãy cho biết trạng thái của bảng sau khi thực hiện chèn lần lượt các khóa 7, 8, 6, 17, 4, 28 vào bảng với $m = 10$