Chương 7: **Các thuật toán tìm kiếm** (SEARCHING)

Nội dung

1. **Bài toán tìm kiếm (searching problem)**
2. Tìm kiếm tuần tự(Linear Search)
3. Tìm kiếm nhị phân (Binary Search)
4. Tìm kiếm trên bảng băm (Hash table)

1. Bài toán tìm kiếm

- Tìm kiếm là một yêu cầu rất thường xuyên trong đời sống hàng ngày cũng như trong tin học
- Ví dụ:
  - Tìm kiếm một sinh viên trong lớp
  - Tìm kiếm một tập tin, thư mục trong máy

- Để đơn giản ta xét bài toán tìm kiếm như sau:
  - Cho một dãy số gồm các phần tử a_1, a_2, ..., a_n. Cho biết trong dãy này có phần tử nào có giá trị bằng X (cho trước) hay không?

Thuật toán tìm kiếm

1. Bài toán tìm kiếm (searching problem)
2. **Tìm kiếm tuần tự (Linear Search)**
3. Tìm kiếm nhị phân (Binary Search)
4. Tìm kiếm trên bảng băm (Hash table)

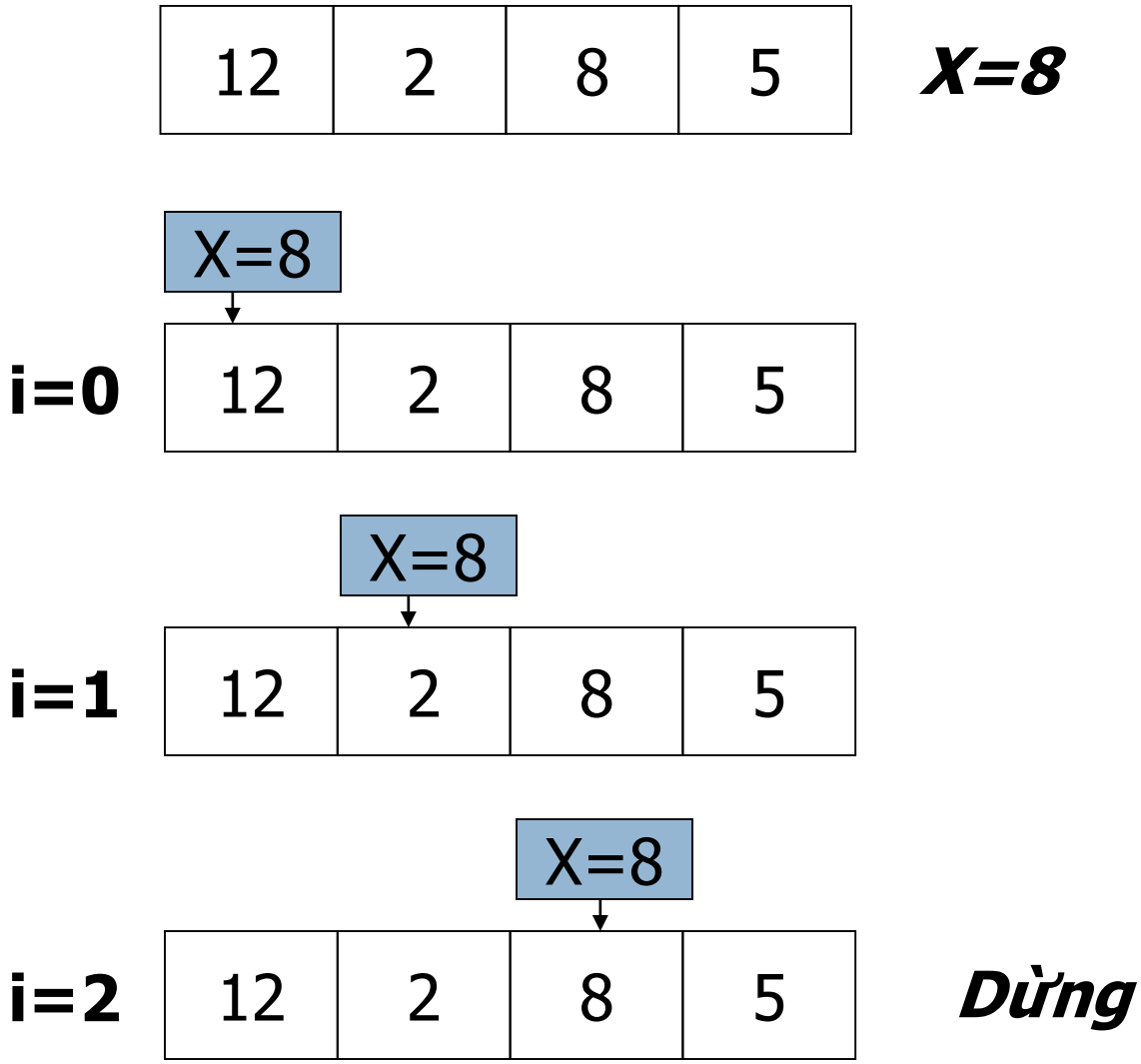
2. Tìm kiếm tuần tự (Linear Seach)

Ý tưởng:

- Bắt đầu từ phần tử đầu tiên của danh sách, so sánh lần lượt từng phần tử của danh sách với giá trị X cần tìm
  - Nếu có phần tử bằng X, thuật toán dừng lại (tìm thấy)
  - Nếu đến cuối danh sách mà không có phần tử nào bằng X, thuật toán dừng lại (không tìm thấy)

2. Tìm kiếm tuần tự (Linear Seach)

Ví dụ 2.1:

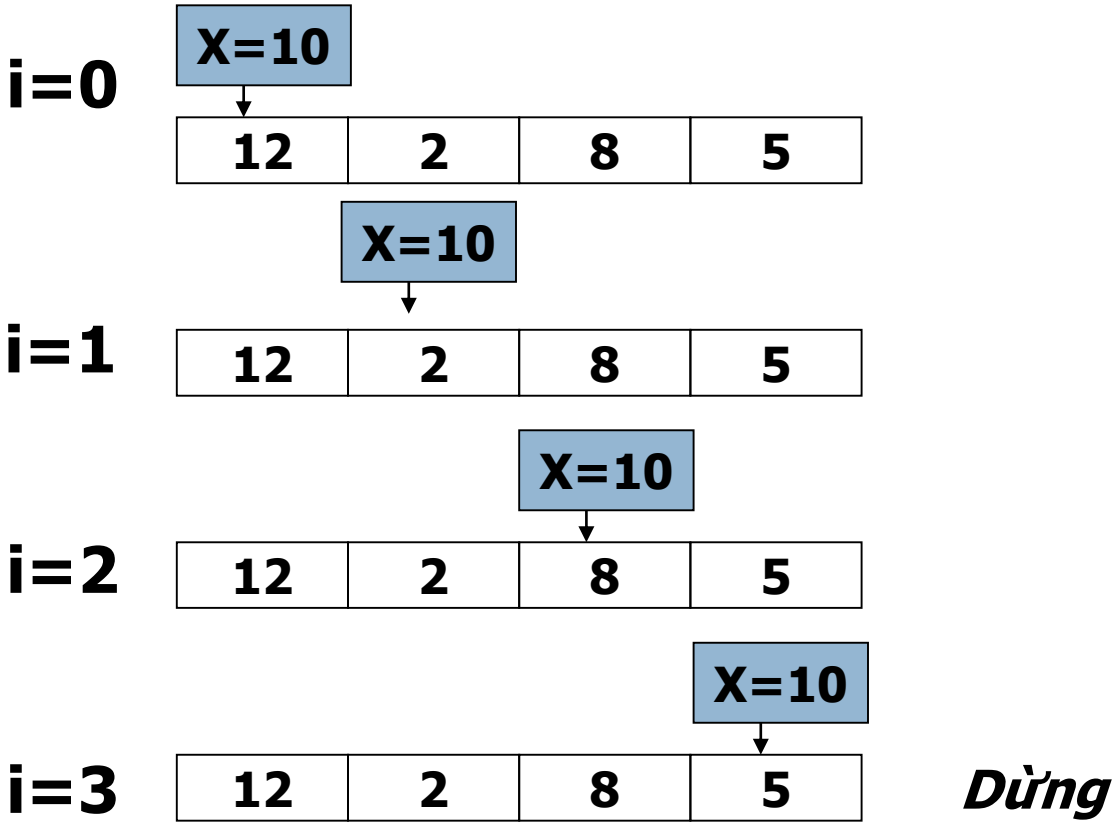


2. Tìm kiếm tuần tự (Linear Seach)

Ví dụ 2.2:

12	2	8	5

X=10



2. Tìm kiếm tuần tự (Linear Seach)

Thuật toán:

B1: i = 0 ; // bắt đầu từ phần tử đầu tiên

B2: so sánh A[i] với X, có 2 khả năng :

- A[i] = X : Tìm thấy. Dừng
- A[i] ≠ X : Sang B3

B3: i=i+1 // Xét phần tử tiếp theo trong mảng

Nếu i=n : Hết mảng, không tìm thấy. Dừng

Ngược lại: lặp lại B2

2. Tìm kiếm tuần tự (Linear Seach)


```
void lsearch (int list[], int n, int key) {
    int flag = 0;          // giả sử lúc đầu chưa tìm thấy
    for(int i=0; i<n; i++)
        if (list[i] == key) {
            cout<<"found at position"<<i;
            flag =1;       // tìm thấy
            break;
        }
    if (flag == 0)
        cout<<"not found";
}
```

2. Tìm kiếm tuần tự (Linear Seach)


```
int lsearch(int list[], int n, int key)
{
   int find= -1;
   for(int i=0; i<n; i++)
         if (list[i] == key)
         {
                  find = i;
                  break;
         }
   return find;
}
```

2. Tìm kiếm tuần tự (Linear Seach)

- Phân tích, đánh giá thuật toán

Trường hợp	Số lần so sánh	Giải thích
Tốt nhất	1	Phần tử đầu tiên có giá trị x
Xấu nhất	n+1	Không có x trong mảng
Trung bình	(n+1)/2	Giả sử xác suất các phần tử trong mảng nhận giá trị x là như nhau.

- Vậy giải thuật tìm tuyến tính có độ phức tạp tính toán cấp n: $T(n) = O(n)$

Thuật toán tìm kiếm

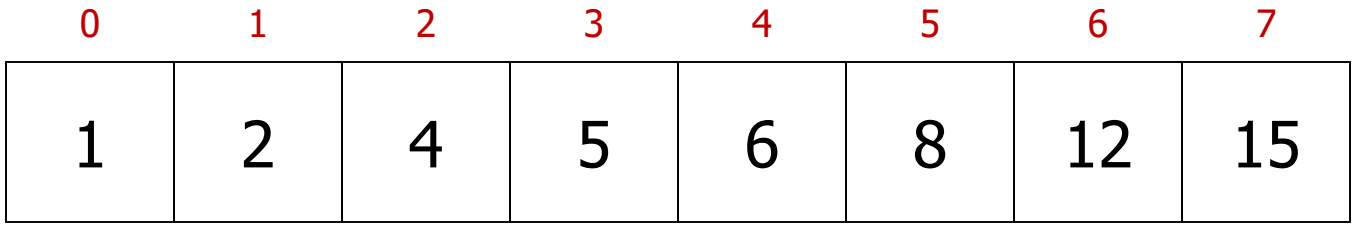
1. Bài toán tìm kiếm (searching problem)
2. Tìm kiếm tuần tự(Linear Search)
3. **Tìm kiếm nhị phân (Binary Search)**
4. Tìm kiếm trên bảng băm (Hash table)

3. Tìm nhị phân (**Binary Seach**)

- Điều kiện:
  - Dữ liệu được lưu bằng mảng và được sắp xếp teo thứ tự của khóa tìm kiếm.
- Ý tưởng:
  - So sánh giá trị muốn tìm X với phần tử nằm ở vị trí giữa của danh sách:
    - Nếu bằng, tìm kiếm dừng lại (thành công)
    - Nếu X lớn hơn thì tiếp tục tìm kiếm ở phần danh sách bên phải phần tử giữa
    - Nếu X nhỏ hơn thì tiếp tục tìm kiếm ở phần danh sách bên trái phần tử giữa

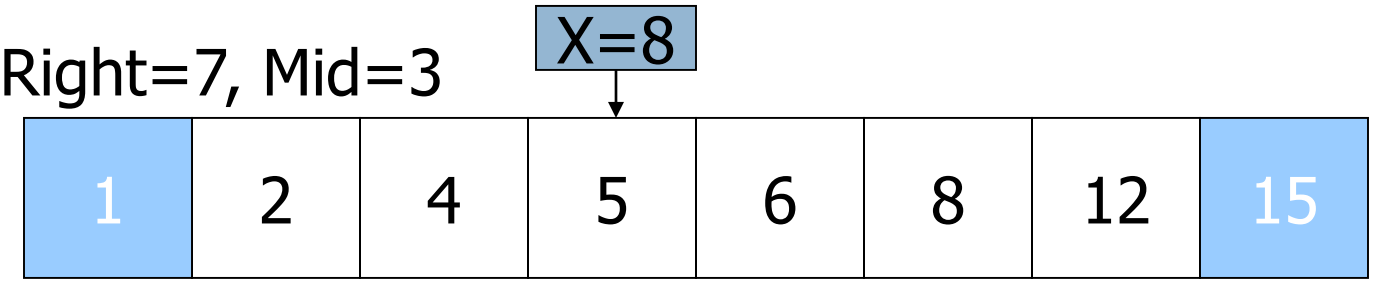
3. Tìm nhị phân (**Binary Seach**)

Ví dụ:

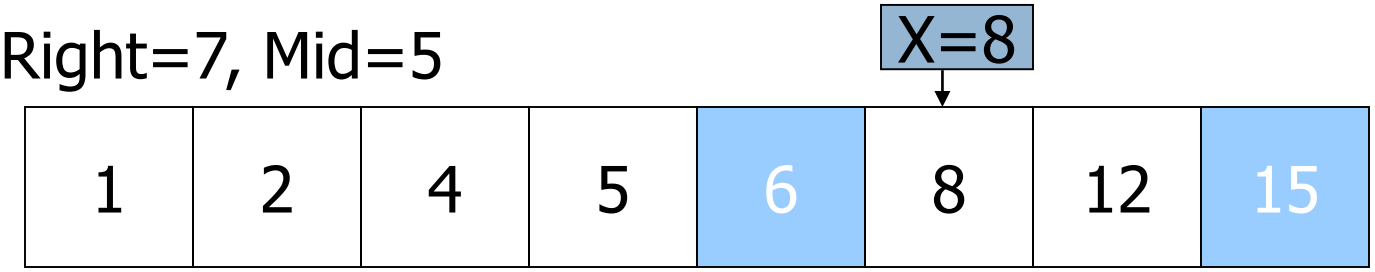


X=8

Left=0, Right=7, Mid=3



Left=4, Right=7, Mid=5



Dừng

3. Tìm nhị phân (**Binary Seach**)

Thuật toán:

B1: Left = 0, Right = n-1

B2: Mid = (Left + Right)/2 // lấy vị trí cận giữa

B3: So sánh X với A[Mid], có 3 khả năng xảy ra:

- A[Mid] = X // tìm thấy. Dừng thuật toán
- A[Mid] > X

  Right = Mid-1 // Tiếp tục tìm trong dãy A[0]… A[Mid-1]
- A[Mid] < X

  Left = Mid+1 // Tiếp tục tìm trong dãy A[Mid+1]… A[Right]

B4: Nếu (Left <= Right) // Còn phần tử chưa xét

Lặp lại B2

Ngược lại: Kết thúc

3. Tìm nhị phân (Binary Seach)

Không đệ quy

```
void BSearch (int list[], int n, int key)
{
    int left, right, mid, flag = 0;
    left = 0; right = n-1;
    while (left <= right) {
            mid = (left + right)/2;
            if( list[mid] == key) {
                    cout<<"found:"<< mid;
                    flag =1;    // đánh dấu tìm thấy
                    break;
            }
            else if (list[mid] < key)   left = mid +1;
            else
                     right = mid -1;
    }
    if (flag == 0)
            cout<<"not found";
}
```

3. Tìm nhị phân (**Binary Seach**)

Đệ quy

```
int BSearch_Recursion (int list[], int key, int left, int right)
{
    if (left <= right)
    {
        int mid = (left + right)/2;
        if (key == list[mid])
            return mid;       // trả về vị trí tìm thấy key
        else if (key < list[mid])
            return BSearch_Recursion (list, key, left, mid-1);
        else return BSearch_Recursion (list, key, mid+1, right);
    }
    return -1;     // không tìm thấy
}
```

3. Tìm nhị phân (**Binary Seach**)

- Phân tích, đánh giá thuật toán:

Trường hợp	Số lần so sánh	Giải thích
Tốt nhất	1	Phần tử giữa của mảng có giá trị x
Xấu nhất	$\log_2 n$	Không có x trong mảng
Trung bình	$\log_2 (n/2)$	Giả sử xác suất các phần tử trong mảng nhận giá trị x là như nhau

- Vậy giải thuật tìm nhị phân có độ phức tạp tính toán cấp n: $T(n) = O(\log_2 n)$

Nhận xét

- Tìm tuần tự không phụ thuộc vào thứ tự của các phần tử, do vậy đây là phương pháp tổng quát nhất để tìm kiếm trên một dãy bất kỳ
- Tìm Nhị Phân dựa vào quan hệ giá trị của các phần tử mảng để định hướng trong quá trình tìm kiếm, do vậy chỉ áp dụng được cho những dãy đã có thứ tự
- Giải thuật Tìm Nhị Phân tiết kiệm thời gian hơn rất nhiều so với giải thuật Tìm tuần tự do:

$$T_{nhị\_phân}(n) = O(\log_2 n) < T_{tuần\_tự}(n) = O(n)$$

Nhận xét

- Tuy nhiên khi muốn áp dụng giải thuật tìm Nhị Phân cần phải xét đến thời gian sắp xếp dãy số để thỏa điều kiện dãy số có thứ tự
- Thời gian này không nhỏ, và khi dãy số biến động cần phải tiến hành sắp xếp lại
- Tất cả các nhu cầu đó tạo ra khuyết điểm chính cho giải thuật tìm Nhị Phân
- Ta cần cân nhắc nhu cầu thực tế để chọn một trong hai giải thuật tìm kiếm trên sao cho có lợi nhất

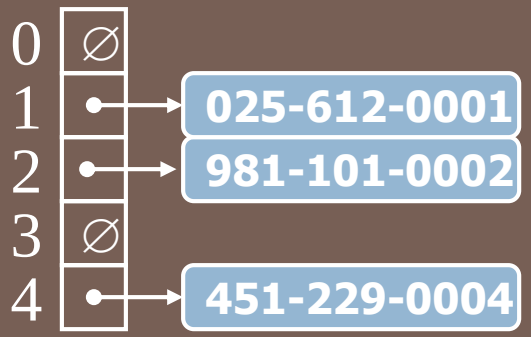
Thuật toán tìm kiếm

1. Bài toán tìm kiếm (searching problem)
2. Tìm kiếm tuần tự(Linear Search)
3. Tìm kiếm nhị phân (Binary Search)
4. **Tìm kiếm trên bảng băm (Hash table)**

TÌM KIẾM THEO ĐỊA CHỈ

***

BẢNG BĂM - HASH TABLES



Bảng băm - Hash table

- Một bảng băm là một cấu trúc dựa trên mảng để lưu trữ các phần tử, mỗi phần tử là một cặp Khóa-Giá trị (key-value). Mỗi phần tử được lưu trữ vào một ô nào đó trong mảng tùy theo khóa của nó là gì
- Các thành phần cấu thành lên bảng băm
  - Mảng chứa
  - Mỗi phần tử mảng quản lý một danh sách các phần tử có khóa qua ánh xạ h cho cùng một địa chỉ.
  - Hàm băm *h(k)* - Hash function, *h*(*k*)
    - Mã băm

Việc hashing được thực hiện qua 2 bước:

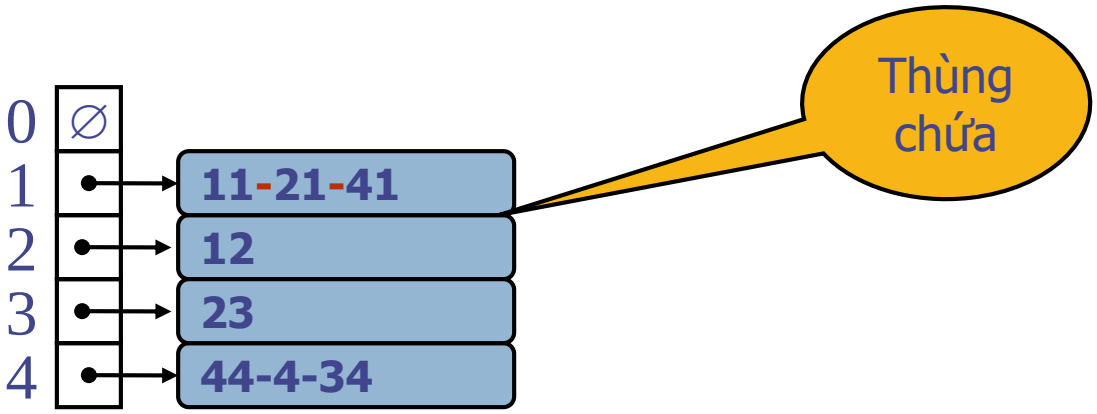
- Một phần tử sẽ được chuyển đổi thành 1 số nguyên bằng việc sử dụng hàm băm. Phần tử này được sử dụng như một chỉ mục để lưu trữ phần tử gốc và nó sẽ được đưa vào bảng băm.
- Phần tử sẽ được lưu giữ trong bảng băm, nó có thể được truy xuất nhanh bằng khóa băm:
  - hash = hashfunc(key)
  - index = hash % array_size

Hàm băm

- Ánh xạ khóa sang số nguyên (vị trí trong bảng băm)
- Nếu nhiều khóa ánh xạ sang cùng một số nguyên (cùng vị trí trong bảng băm) sẽ dẫn đến **đụng độ**

Giả sử có hàm h(k) = k % 5

Có các giá trị: 11, 21, 44, 23, 41, 4, 34, 12



Để giảm đụng độ, ta thường chọn kích thước bảng là một số nguyên tố, thông thường dạng 4k+3

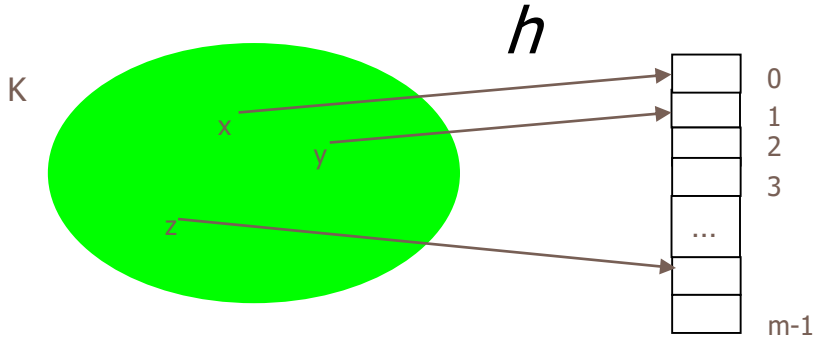
Hàm băm

Cấu trúc hàm băm

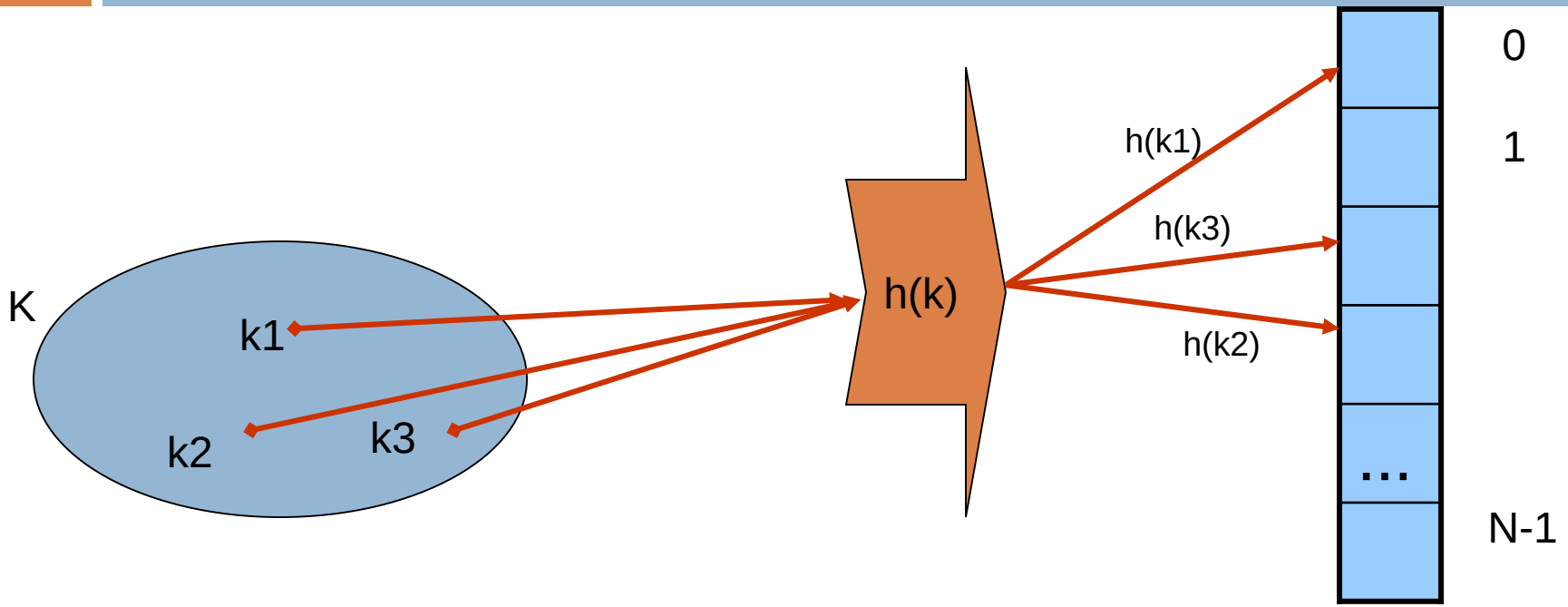
- Hàm băm có dạng như sau:

$$h : K \rightarrow 0..m\text{-}1$$

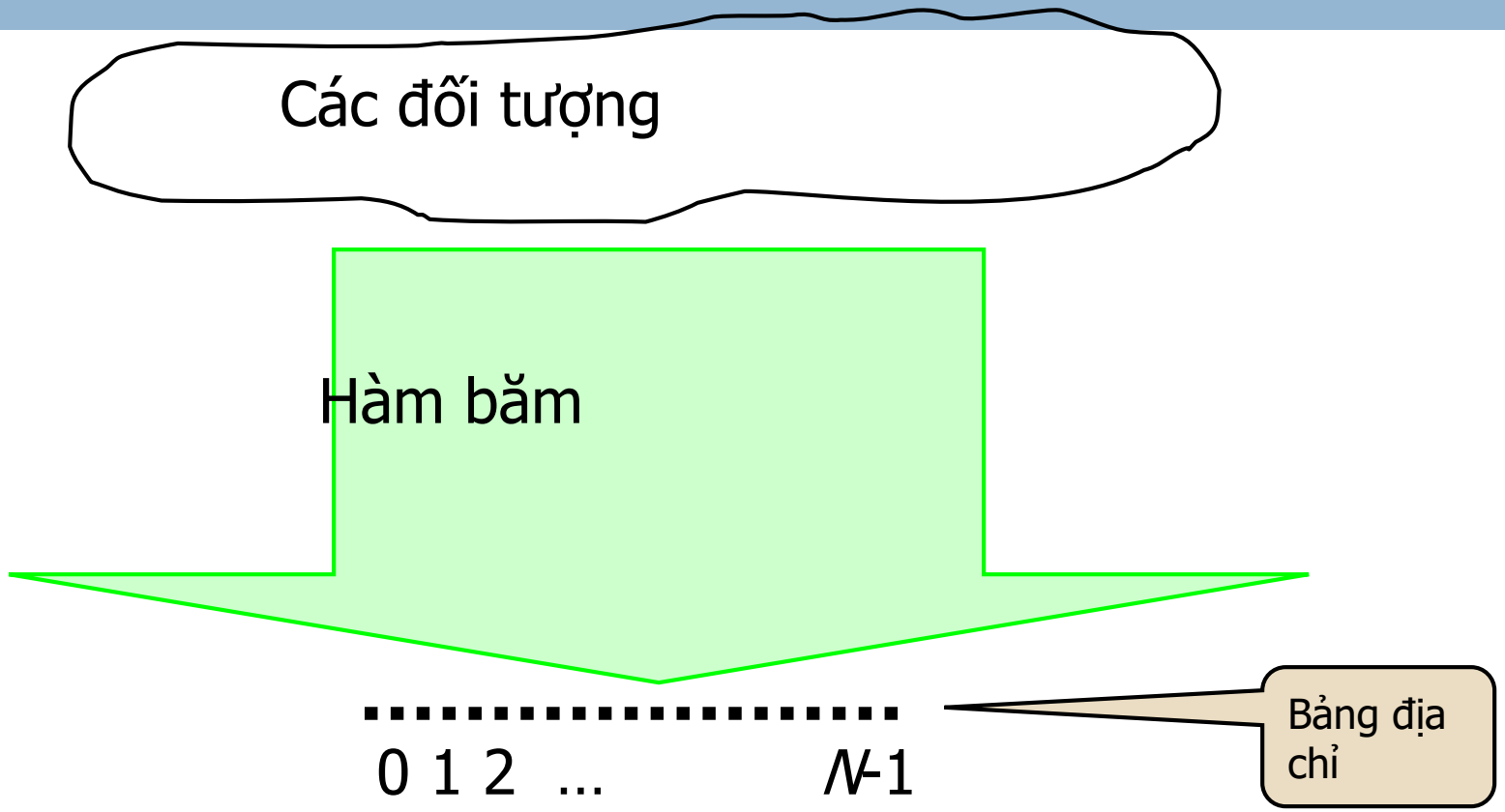
- Trong đó:
  - *h* được gọi là hàm băm (hash function)
  - K là tập giá trị khóa
  - 0..m-1 là bảng địa chỉ (là các số nguyên)
  - m là kích thước của bảng (tablesize)
- Yêu cầu khi xây dựng hàm băm:
  - Hàm phải dải đều các địa chỉ trên bảng địa chỉ
  - Hàm băm phải được tính toán đơn giản.



Hình ảnh hàm băm



Hàm băm



Một hàm băm ánh xạ các đối tượng vào tập các địa chỉ [0, N-1]

Hàm băm

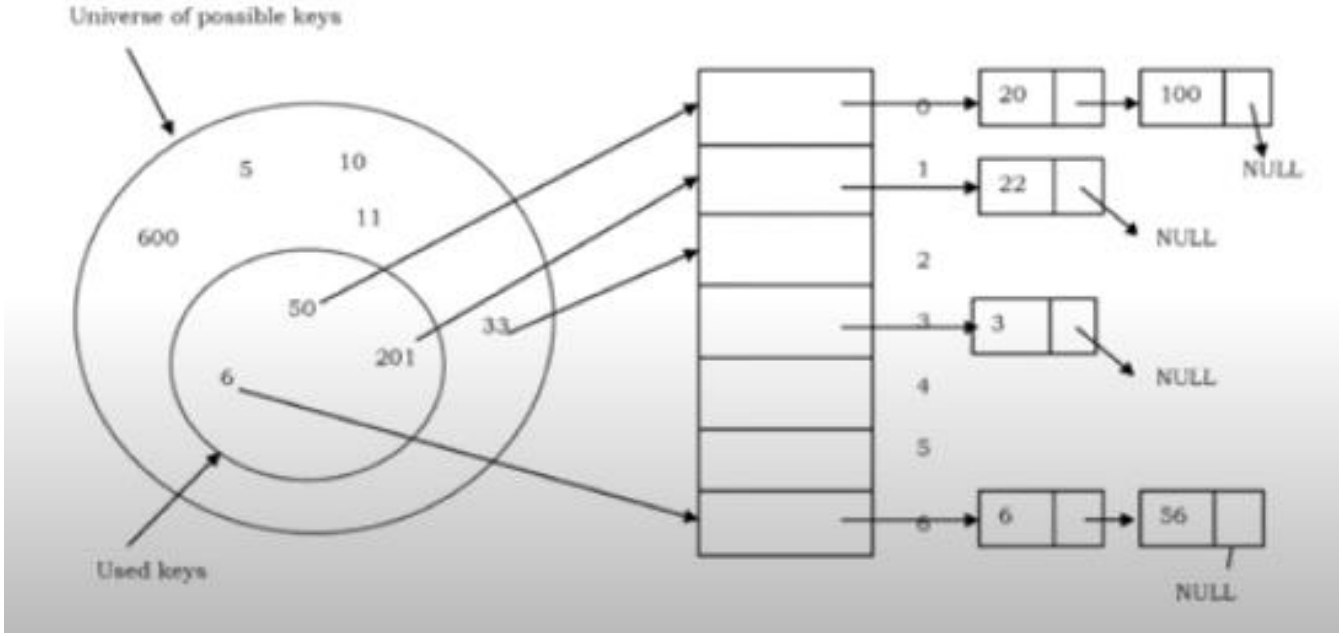
- Một hàm băm đơn giản dùng phép chia lấy dư:
- h(k)=k%m (m là tablesize)
- Giả sử:
- k=24, 48, 51, 78, 15
- m=10
- → k%10=4, 8, 1, 8, 5

9	
8	
7	
6	
5	
4	
3	
2	
1	
0	

Một số phương pháp xây dựng hàm băm

- Phương pháp dây chuyền
- Phương pháp thăm dò

- Phương pháp dây chuyền: sử dụng con trỏ và danh sách liên kết



Bảng thăm dò

- Bảng băm dây chuyền phức tạp do phải duy trì một danh sách liên kết ở mỗi ô
- Giải pháp thăm dò ô trống:
  - Nếu đụng độ xảy ra, thử các ô khác trong bảng
  - Thử lần lượt các ô cho đến khi tìm được 1 ô trống
    - h(k)=(hash(k)+f(i))%tablesize

Thăm dò tuyến tính

- h(k)=(hash(k)+f(i))%tablesize (f(i)=i là số lần băm lại, hash(k) là giá trị thu được trong lần băm trước đó)

- Phép chèn (giả sử các khóa không trùng nhau)

1. Index=hash(k)
2. 2. Nếu table[index] rỗng, đặt phần tử mới vào

3. Nếu table[index] không rỗng

- Index ++; index=index%tablesize
- Quay lại bước 2

- Tìm kiếm

1. Index=hash(k)
2. Nếu table[index] rỗng, trả về -1 (không tìm thấy)
3. Nếu table[index] ==k, trả về index (tìm thấy)
4. Index++; index=index%tablesize; quay lại bước 2

Ví dụ:

- Chèn 89, 18, 49, 58, 69 vào bảng băm
- h(k)=k%10

	Empty	89	18	49	58	69
0						
1						
2						
3						
4						
5						
6						
7						
8						
9						

- Hash function h(key) = key % 10
- Key: 32,53,22,93,34,17

Thăm dò bậc 2

- h(k)=(hash(k)+f(i))%tablesize, f(i)=i^2 với i là số lần băm lại (băm lại lần 1 là 1^2, nếu còn đụng độ cho băm lần 2 là 2^2, …)

	Empty	89	18	49	58	69
0						
1						
2						
3						
4						
5						
6						
7						
8						
9						

Phương pháp chia

- Để tính địa chỉ dải của đối tượng ta lấy giá trị khóa chia cho kích thước của bảng. Địa chỉ dải là phần dư của phép chia đó.

$$h(k) = k \% m$$

- Yêu cầu:
  - Hàm h phải dải đều các đối tượng trên bảng một cách ngẫu nhiên. Để có được điều đó h phải phụ thuộc vào m.
  - Thông thường người ta chọn m là một số nguyên tố nhỏ hơn gần với (10, 100, 1000,...) nhất.

Phương pháp nhân

- Giá trị khóa được nhân với chính nó sau đó lấy con số bao gồm một số chữ số ở "giữa" kết quả để làm "địa chỉ rải".
- Ví dụ:

k	k^2	h(k) gồm 3 chữ số
5402	29181604	181 hoặc 816
0367	00134689	134 hoặc 346
1246	01552516	552 hoặc 525

◆ Rõ ràng các chữ số ở giữa phụ thuộc vào mọi chữ số của khóa do đó nếu khóa có khác nhau chút ít thì địa chỉ dải vẫn khác nhau nhiều.

Phương pháp chia

- Phương pháp chia
- h(k) = k mod SIZE
- nhạy cảm với cỡ của bảng băm
- chọn SIZE để hạn chế xảy ra va chạm
- số nguyên tố có dạng đặc biệt, chẳng hạn có dạng 4k+3

Phương pháp nhân

- $h(k) = \lfloor (\alpha k - \lfloor \alpha k \rfloor) \, . \, SIZE \rfloor$
- Thực tế thường chọn

$$\alpha = \Phi^{-1} \approx 0{,}61803399$$

Cấu trúc dữ liệu bảng băm

- Thuộc tính
  - Mảng (mỗi phần tử mảng lưu một danh sách các phần tử)
  - N: kích thước mảng
- Các phương thức
  - Node *Add(Key, Object)
  - void Remove(Key)
  - Node *Find(Key)
  - bool Contains(Key)
  - int Count()

Độ phức tạp

- Với bảng băm thực hiện các phép tìm/chèn/xóa trong thời gian O(1)
- Không hiệu quả với các thao tác đòi hỏi thông tin thứ tự: sắp xếp, tìm phần tử lớn nhất, nhỏ nhất

→ **So sánh các cấu trúc:**

- Vector, dslk: O(n)
- Cây AVL: O(logN)
- Bảng băm: O(1)

Bài tập

1. Viết chương trình nhập vào một dãy số nguyên. Xây dựng hàm băm để tìm kiếm một phần tử nhập từ bàn phím. *(hàm băm theo số)*

2. Viết chương trình tìm kiếm một sinh viên (sử dụng lớp sinh viên đã học) sử dụng bảng băm *(hàm băm theo chữ)*.

Bài tập

1. Hãy vẽ bảng băm 11 ô, dùng hàm băm h(i) = (3i+5) mod 11. Lần lượt thêm các khóa 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5. Xét 2 phương án:
   a) Giải quyết va chạm bằng phương pháp thăm dò tuyến tính.
   b) Giải quyết va chạm bằng phương pháp thăm dò bình phương.

2. Xét việc lưu tập hợp U các phần tử có giá trị khóa thuộc tập $\{0, 1, \ldots, n^2-1\}$ trong bảng băm. Với từng phương pháp giải quyết va chạm nêu dưới, một bảng băm cỡ n có thể lưu tối đa bao nhiêu khóa phân biệt?
   a) Thăm dò tuyến tính
   b) Thăm dò bình phương
   c) Tạo dây chuyền

Bài tập bảng băm

- Cho dãy các từ khóa: **72, 10, 43, 36, 5, 6, 15, 53, 20, 18**
- Hash table size= 10
- Hash function = ?
- Lưu trữ vào bảng băm các khóa trên theo 2 trường hợp:
  - Sử dụng phương pháp kết nối để xử lý khi có đụng độ
  - Sử dụng phương pháp đo tuyến tính để xử lý khi có đụng độ

Bài tập tổng hợp

- Sinh viên cài đặt thuật toán tìm kiếm tuần tự, tìm kiếm tuyến tính, tìm kiếm trên bảng băm trên một dãy số nguyên được nhập vào từ bàn phím. (Sử dụng ngôn ngữ lập trình C++)