Cấu trúc dữ liệu và giải thuật

CÁC CHIẾN LƯỢC TÌM KIẾM

Giảng viên:
Văn Chí Nam – Nguyễn Thị Hồng Nhung – Đặng Nguyễn Đức Tiến

Nội dung trình bày

- Giới thiệu
- Tìm kiếm tuần tự
- Tìm kiếm nhị phân
- Tổng kết

Giới thiệu

- Thao tác tìm kiếm rất phổ biến trong cuộc sống hàng ngày.
  - Tìm kiếm hồ sơ, tập tin.
  - Tìm kiếm tên người trong danh sách.
  - …

Thuật toán tìm kiếm

- Có nhiều loại:
  - Tìm kiếm tuần tự (Sequential/ Linear Search)
  - Tìm kiếm nhị phân (Binary Search)
  - …
- Mục tiêu:
  - Tìm hiểu về 2 thuật toán tìm kiếm cơ bản.
  - Phân tích thuật toán để lựa chọn thuật toán phù hợp khi áp dụng vào thực tế.

Tìm kiếm tuần tự

Sequential Search

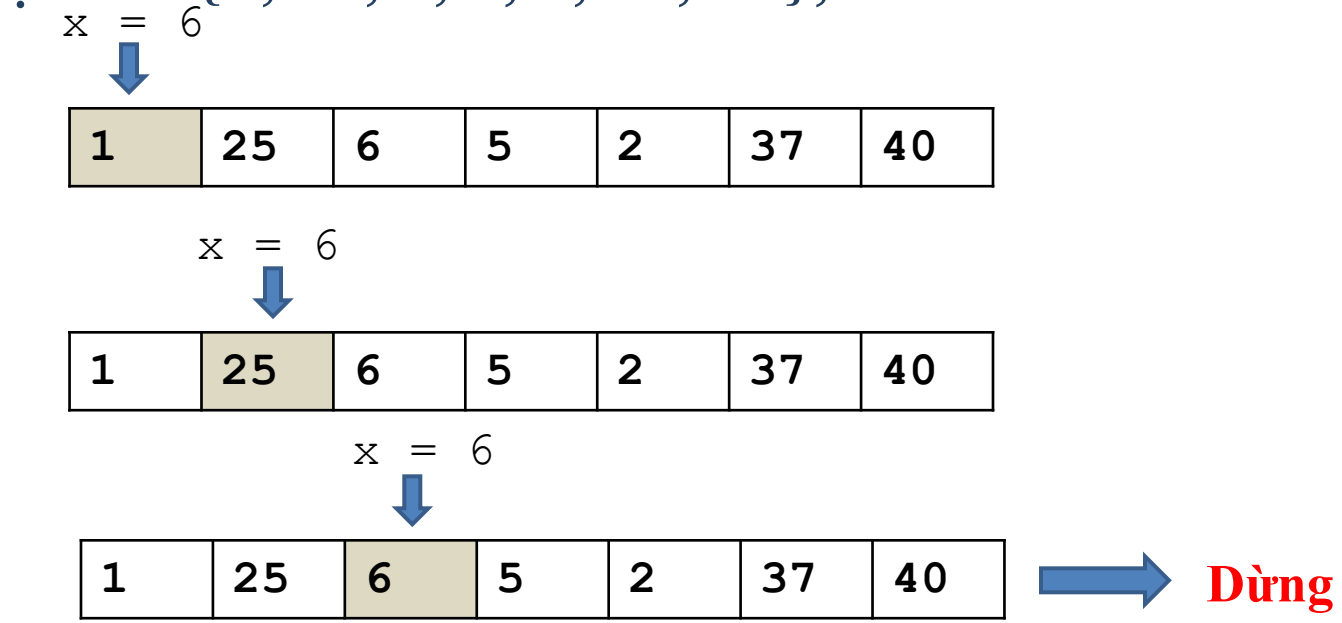
Linear Search

Thuật toán tìm kiếm tuần tự

- Input:
  - Dãy A, *n* phần tử
  - Giá trị *x* cần tìm
- Output:
  - Nếu *x* xuất hiện trong A: trả về vị trí xuất hiện đầu tiên của *x*
  - Nếu không: trả về n hoặc -1
- Thuật toán:
  - Vét cạn (exhaustive)
  - Dùng lính canh (sentinel)

Tìm kiếm tuần tự - Vét cạn

- Thuật toán:
  - Lần lượt so sánh x với các phần tử của mảng A cho đến khi gặp được phần tử cần tìm, hoặc hết mảng.
  - Ví dụ: A = {1, 25, 6, 5, 2, 37, 40}, x = 6



Tìm kiếm tuần tự - Vét cạn

Thuật toán: **LinearExhaustive**

- Bước 1. Khởi tạo biến chỉ số: *i = 0*
- Bước 2. Kiểm tra xem có thực hiện hết mảng hay chưa: *So sánh i và n*
  - Nếu chưa hết mảng (i < n), sang bước 3.
  - Nếu đã hết mảng (i >= n), thông báo không tìm thấy giá trị x cần tìm.
- Bước 3. *So sánh giá trị a[i] với giá trị x* cần tìm
  - Nếu a[i] bằng x: Kết thúc chương trình và thông báo đã tìm thấy x.
  - Nếu a[i] khác x, *tăng i thêm 1* và quay lại bước 2.

Tìm kiếm tuần tự - Vét cạn

- Nhận xét: Phép so sánh là phép toán sơ cấp được dùng trong thuật toán. Suy ra, số lượng các phép so sánh sẽ là thước đo độ phức tạp của thuật toán.

- Mỗi vòng lặp có 2 điều kiện cần kiểm tra:
  - Kiểm tra cuối mảng (bước 2)
  - Kiểm tra phần tử hiện tại có bằng *x*? (bước 3)

Tìm kiếm tuần tự - Vét cạn

- Trường hợp x nằm ở 2 biên của mảng A: rất hiếm khi xuất hiện.
- Ước lượng số vòng lặp trung bình sẽ hữu ích hơn.
- Số phép so sánh trung bình:

$$2(1+2+ \ldots + n)/n = n+1$$

=> Số phép so sánh tăng/giảm tuyến tính theo số phần tử

Tìm kiếm tuần tự - Vét cạn

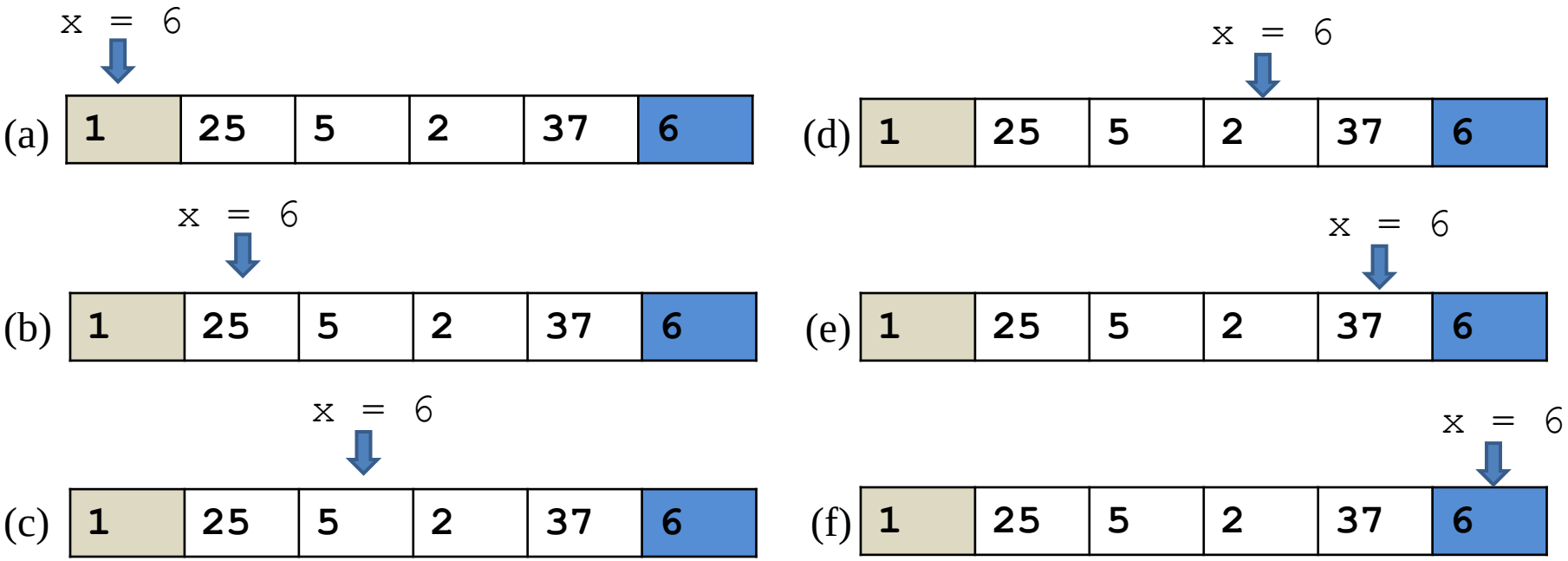
- Vậy độ phức tạp của thuật toán là:
  - Tốt nhất: O(1).
  - Trung bình: O(n).
  - Xấu nhất: O(n).

Tìm kiếm tuần tự - Lính canh

- Trong thuật toán vét cạn, có 2 điều kiện được kiểm tra.
- Có thể bỏ việc kiểm tra điều kiện cuối mảng bằng cách dùng “lính canh”.
- Lính canh là phần tử có giá trị bằng với phần tử cần tìm và đặt ở cuối mảng.

Tìm kiếm tuần tự - Lính canh

- Ví dụ: A = {1, 25, 5, 2, 37}, x = 6



return 5;

Tìm kiếm tuần tự - Lính canh

Thuật toán: **LinearSentinel**

- Bước 1. Khởi tạo biến chỉ số: *i = 0*
- Bước 2. *So sánh giá trị a[i] với giá trị x* cần tìm
  - Nếu a[i] bằng x:
    - Nếu i < n: Kết thúc chương trình và thông báo đã tìm thấy x.
    - Nếu i >= n: Thông báo không tìm thấy x trong mảng.
  - Nếu a[i] khác x, *tăng i thêm 1* và quay lại bước 2.

Tìm kiếm tuần tự - Lính canh

- Thực nghiệm cho thấy trong trường hợp n lớn, thời gian tìm kiếm giảm khi dùng phương pháp lính canh.
  - Với n =15000: nhanh hơn khoảng 20% (0.22s so với 0.28s)

Tìm kiếm nhị phân

Binary Search

Thuật toán tìm kiếm nhị phân

- Với dãy A được sắp xếp thứ tự (ví dụ: tăng dần), độ phức tạp của thuật toán tìm kiếm tuần tự không đổi.

- Tận dụng thông tin của mảng đã được sắp xếp để giới hạn vị trí của giá trị cần tìm trong mảng.

-> Thuật toán tìm kiếm nhị phân.

Thuật toán tìm kiếm nhị phân

- Input:
  - Dãy A, *n* phần tử **đã được sắp xếp**
  - Giá trị *x* cần tìm
- Output:
  - Nếu *x* xuất hiện trong A: trả về một vị trí xuất hiện của *x*
  - Nếu không: trả về n hoặc -1

Thuật toán tìm kiếm nhị phân

- Ý tưởng:
  - So sánh x với phần tử chính giữa mảng A.
    - Nếu x là phần tử giữa thì dừng.
  - Nếu không: xác định xem ***x*** có thể thuộc nửa trái hay nửa phải của A.
  - Lặp lại 2 bước trên với nửa đã được xác định.

Thuật toán tìm kiếm nhị phân

Thuật toán: BinarySearch(A[], n, x)

- Bước 1. Khởi gán left = 0 và right = n – 1.
- Bước 2. Trong khi left <= right, thực hiện:
  - 2.1. Đặt *mid = (left + right)/2*
  - 2.2. *So sánh giá trị x và a[mid]*:
    - Nếu x < a[mid], gán *right = mid – 1*.
    - Nếu x > a[mid], gán *left = mid + 1*.
    - Nếu x = a[mid], thông báo đã tìm thấy x và kết thúc.
- Kết quả trả về không tìm thấy x nếu left > right*.

** Điều này có nghĩa là không còn phần tử nào trong mảng: x không có trong mảng*

Thuật toán tìm kiếm nhị phân

Cài đặt đệ quy: BinarySearch(A[], left, right, x)

- Bước 1. Nếu left > right: thông báo không tìm thấy x và thoát khỏi hàm.
- Bước 2.
  - 2.1. Đặt *mid = (left + right)/2*
  - 2.2. *So sánh giá trị x và a[mid]*:
    - Nếu x < a[mid], Gọi BinarySearch(A, left, mid – 1, x)
    - Nếu x > a[mid], Gọi BinarySearch(A, mid + 1, right, x)
    - Nếu x = a[mid], thông báo đã tìm thấy x và kết thúc (trả lại giá trị mid)

Thuật toán tìm kiếm nhị phân

- Minh họa:
  - A[] = {1, 2, 6, 26, 28, 37, 40}, x = 2

index	0	1	2	3	4	5	6
A[i]	1	2	6	26	28	37	40
Vòng 1	**left**			**mid**			**right**
Vòng 2	**left**	**mid**	**right**				

```
x = a[1] -> return 1
```

Thuật toán tìm kiếm nhị phân

- Minh họa:
  - A[] = {1, 2, 6, 26, 28, 37, 40}, x = 40

index	0	1	2	3	4	5	6
A[i]	1	2	6	26	28	37	40
Vòng 1	**left**			**mid**			**right**
Vòng 2					**left**	**mid**	**right**
Vòng 3							**left** **mid** **right**

```
x = a[6] -> return 6
```

Thuật toán tìm kiếm nhị phân

- Minh họa:
  - A[] = {1, 2, 6, 26, 28, 37, 40}, x = -7

index	0	1	2	3	4	5	6
A[i]	1	2	6	26	28	37	40
Vòng 1	**left**			**mid**			**right**
Vòng 2	**left**	**mid**	**right**				
Vòng 3	**left** **mid** **right**						
Vòng 4							

right = -1, left = 0

=> **right < left** => thoát khỏi while,

return -1

Thuật toán tìm kiếm nhị phân

- Phân tích thuật toán tuyến tính:
  - Mỗi lần lặp thì chiều dài của mảng con giảm *khoảng* ½ so với mảng trước đó.
  - $n = 2^k + m \ (0 \le m < 2)$
  - $2^k \le n < 2^{k+1} \Rightarrow k \le \log_2 n < k+1 \Rightarrow k = \lfloor \log_2 n \rfloor$

  => mảng A ban đầu được chia nửa *khoảng* **k** lần.
  - Số lần thực hiện vòng while là khoảng k lần, mỗi vòng lặp thực hiện 1 phép so sánh.

Thuật toán tìm kiếm nhị phân

- Phân tích thuật toán tuyến tính:
  - Trường hợp tốt nhất: k = 1 ⇔ x là phần tử chính giữa của mảng.
  - Trường hợp xấu nhất: $k = \lfloor \log_2 n \rfloor + 1$ ⇔ x không thuộc mảng hoặc x là phần tử cuối cùng của mảng

  => Số phép so sánh tăng theo hàm logarit

Thuật toán tìm kiếm nhị phân

- Độ phức tạp của tìm kiếm nhị phân
  - Trường hợp tốt nhất: O(1)
  - Trường hợp trung bình: $O(\log_2 n)$
  - Trường hợp xấu nhất: $O(\log_2 n)$

So sánh hiệu suất

- So sánh trường hợp xấu nhất của 2 thuật toán:

Kích thước mảng	Trường hợp xấu nhất	
	Tuần tự	Nhị phân
100.000	100.000	16
200.000	200.000	17
400.000	400.000	18
800.000	800.000	19
1.600.000	1.600.000	20

Tổng kết

- Có nhiều thuật toán tìm kiếm, ước lượng số phép so sánh của mỗi thuật toán cho biết hiệu suất của thuật toán.
- Thuật toán tuần tự tìm kiếm cho đến khi tìm thấy giá trị cần tìm hoặc hết mảng
- Hiệu suất của tìm kiếm tuần tự trong trường hợp xấu nhất là 1 hàm tuyến tính theo số phần tử mảng.

Tổng kết

- Nếu mảng đã được sắp xếp thì nên dùng tìm kiếm nhị phân.
- Tìm kiếm nhị phân dùng kết quả của phép so sánh để thu hẹp vùng tìm kiếm kế tiếp.
- Hiệu suất của tìm kiếm nhị phân là một hàm logarit theo số phần tử mảng.

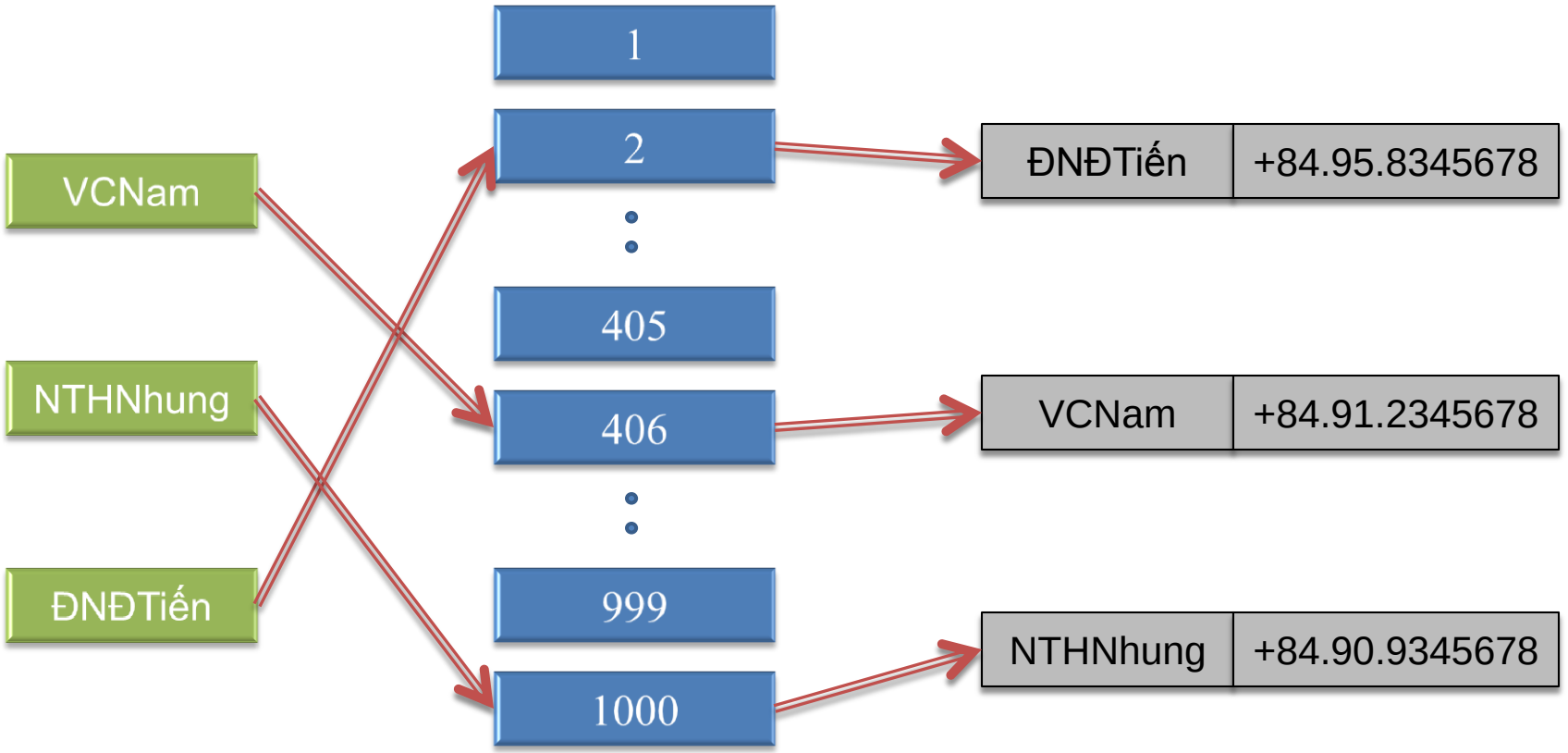
Tìm kiếm theo bảng băm

Hash Table

Khái quát về hash

- ***Vấn đề:*** Cho trước 1 tập S gồm các phần tử được đặc trưng bởi giá trị khóa. Trên giá trị các khóa này có quan hệ thứ tự. Tổ chức S như thế nào để tìm kiếm 1 phần tử có khóa k cho trước có độ phức tạp ít nhất trong giới hạn bộ nhớ cho phép?
- ***Ý tưởng:*** Biến đổi khóa k thành một số (bằng hàm hash) và sử dụng số này như là địa chỉ để tìm kiếm trên bảng dữ liệu.

Ví dụ về một bảng băm



Độ phức tạp

- Chi phí tìm kiếm trung bình: O(1)
- Chi phí tìm kiếm trong trường hợp xấu nhất: O(n) (rất ít gặp).

Hàm băm

- ***Định nghĩa:*** Hàm băm (hash function) là hàm biến đổi khóa k của phần tử thành địa chỉ trong *bảng băm*.

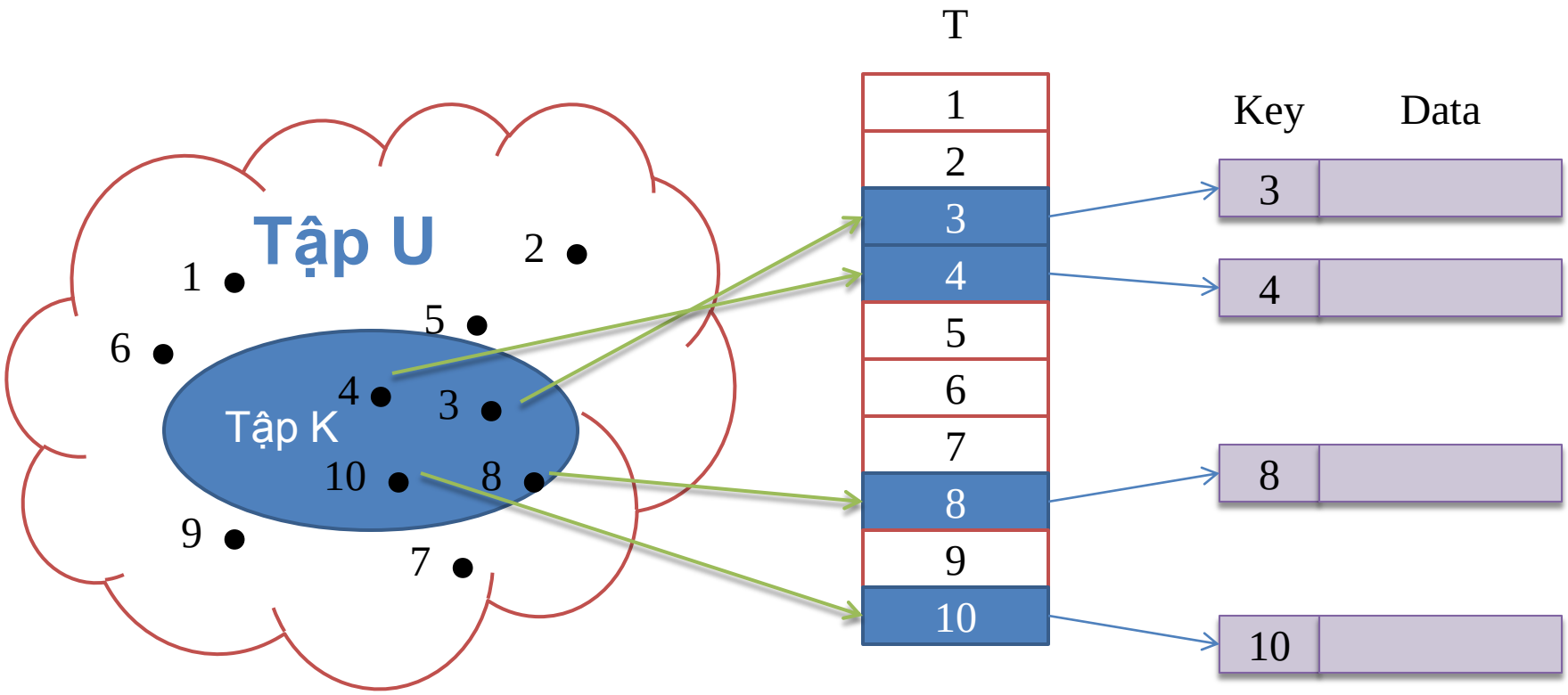
- ***Tổng quát về phép biến đổi khóa:*** Là 1 ánh xạ thích hợp từ tập các khóa U vào tập các địa chỉ A.

$$H: U \rightarrow A$$

$$k \rightarrow a = h(k)$$

- Tập các giá trị khóa (U) có thể lớn hơn rất nhiều so với số khóa thực tế (K) rất nhiều.

Hàm băm



Hàm băm

- Chọn số (Digit-selection):
  - Chọn một vài chữ số trong khóa và ghép lại tạo thành giá trị băm.
  - Ví dụ:
    - h(001364825) = 35
  - Ưu điểm: Đơn giản, tính toán nhanh
  - Khuyết điểm: Không thể hiện tính chất của khóa, không phân bố đều

Hàm băm

- Gấp số (folding)
  - Cộng các chữ số của khóa
  - Nhóm các chữ số thành số và cộng lại
  - Ví dụ:
    - h(001364825) = 0 + 0 + 1 + 3 + 6 + 4 + 8 + 2 + 5 = 29
    - h(**001**364**825**) = 001 + 364 + 825 = 1190

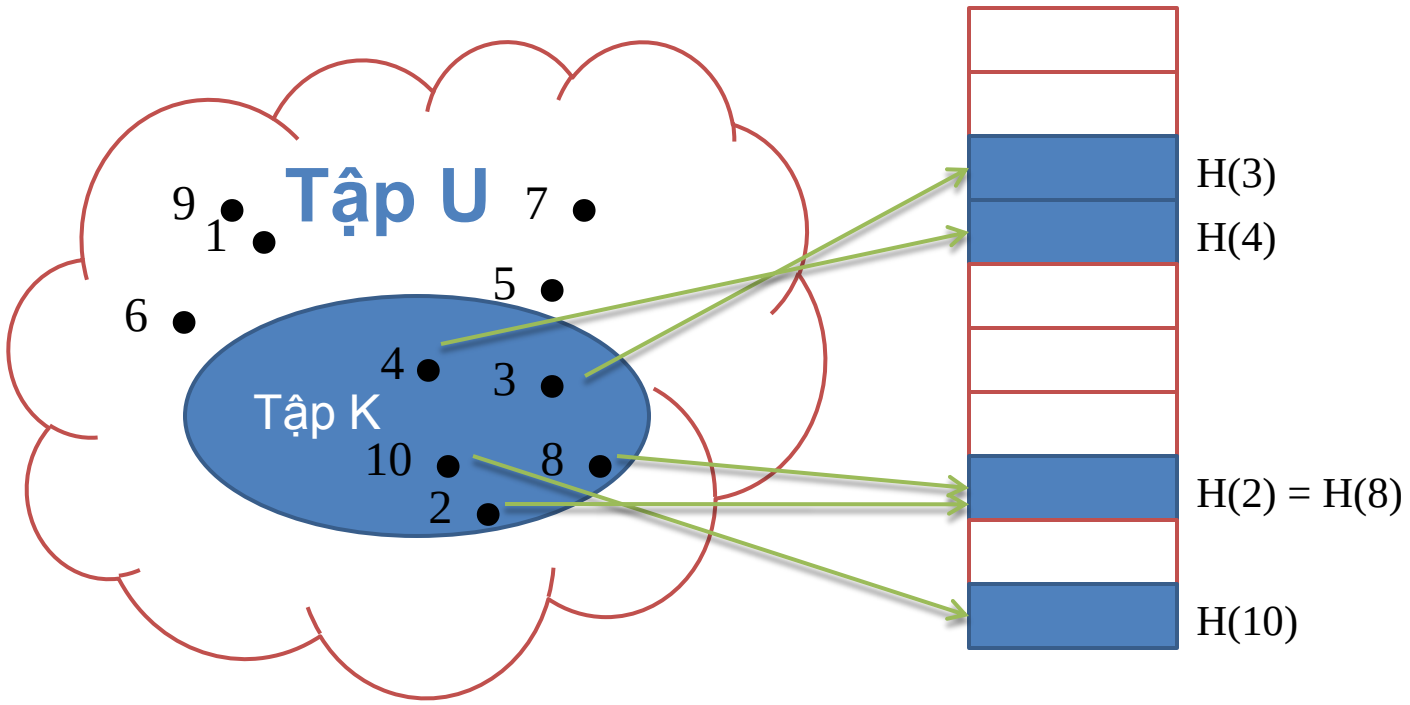
Hàm băm

- Lấy dư (modulo arithmetic)
  - Sử dụng phép tính lấy dư
  - h (Key) = Key **mod** tableSize
  - Ví dụ:
    - h(Key) = Key mod 101
    - h(001364825) = 12

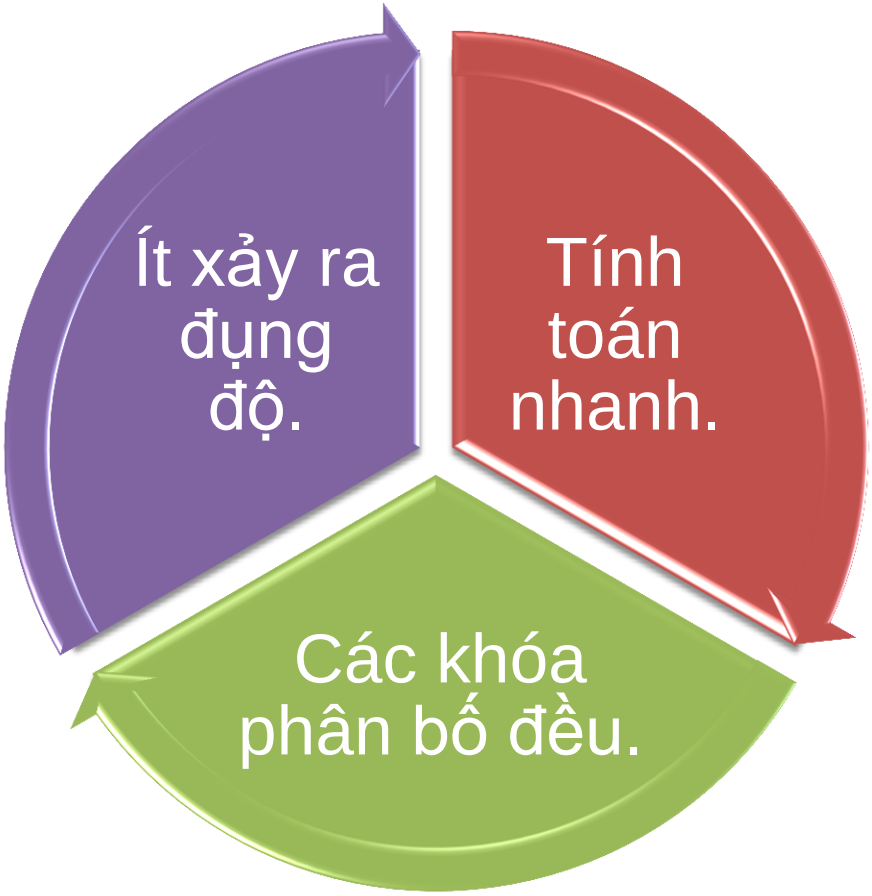
Sự đụng độ (collision)

- $\exists k_1, k_2 \in K$:

$$k_1 \neq k_2, H(k_1) = H(k_2)$$



Những yêu cầu đối với hàm băm



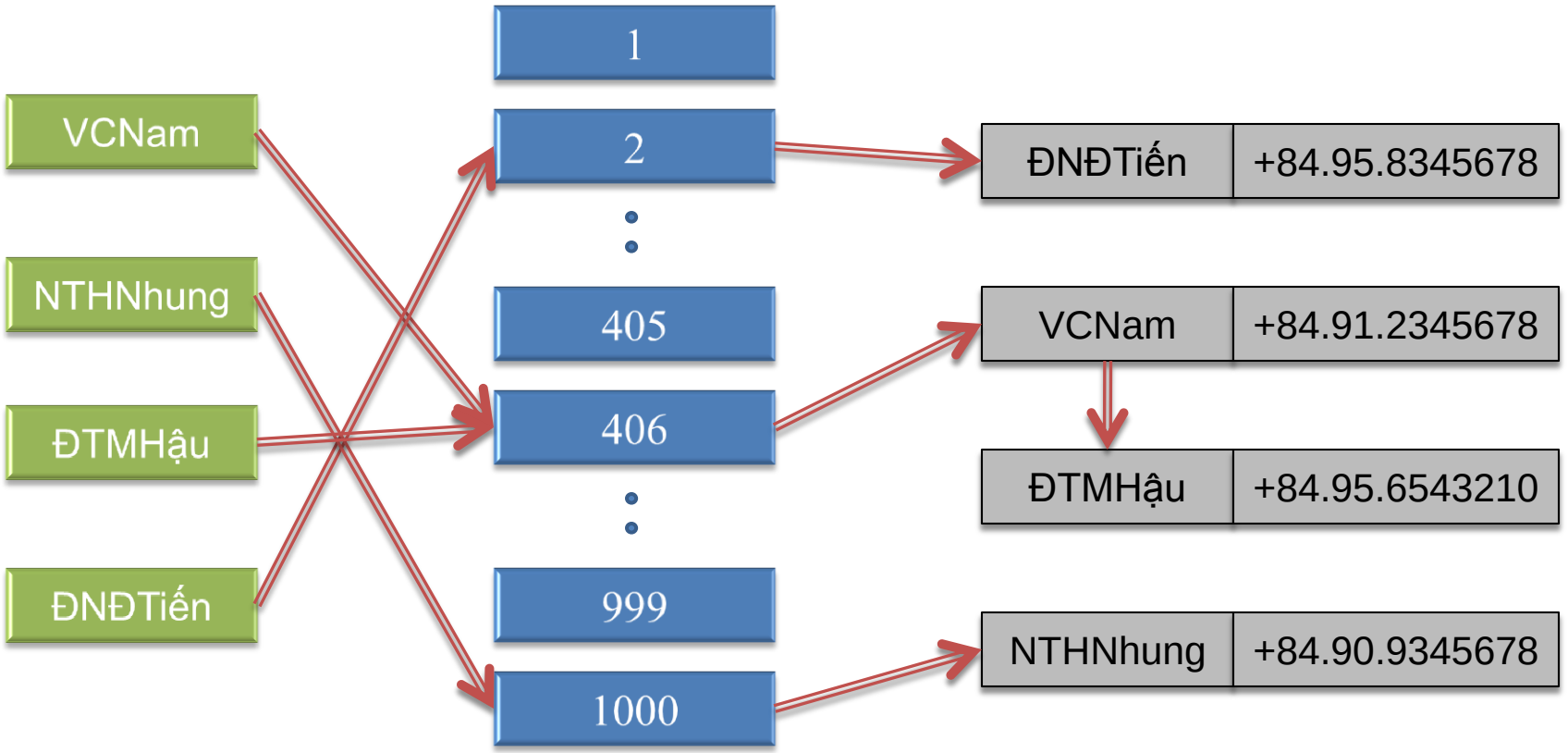
Các phương pháp xử lý đụng độ

- Phương pháp nối kết (separate chaining)
- Phương pháp địa chỉ mở (Open-addressing)

Phương pháp nối kết

- Ứng với mỗi địa chỉ của bảng, ta có một danh sách liên kết chứa các phần tử có khóa khác nhau mà có cùng địa chỉ đó.
- Ta sẽ có danh sách (bảng băm) gồm M phần tử chứa địa chỉ đầu của các danh sách liên kết.

Phương pháp nối kết



Phương pháp địa chỉ mở

- Tên gọi khác:
  - Phương pháp dò
  - Phương pháp thử
- Ý tưởng:
  - Khi đụng độ xảy ra, ta sẽ thử tìm đến vị trị kế tiếp nào đó trong bảng cho đến khi tìm thấy vị trí nào còn trống.

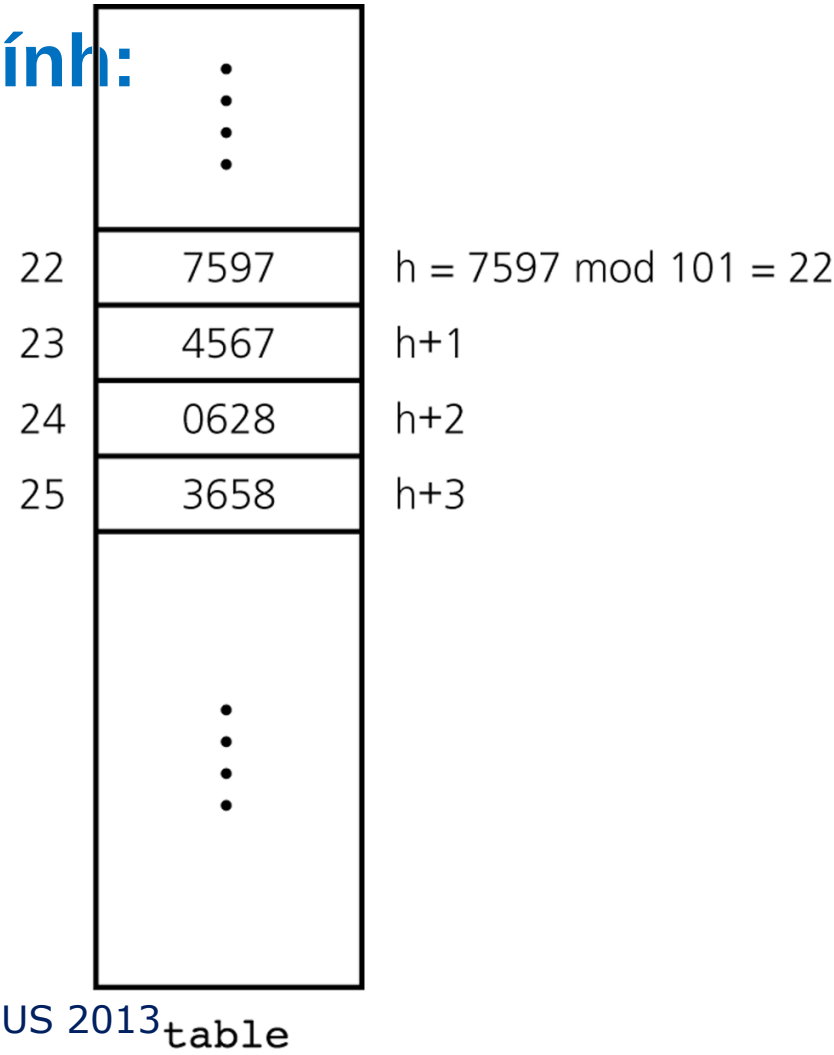
Phương pháp địa chỉ mở

- Phương pháp dò tuyến tính (Linear probing)
- Phương pháp dò bậc 2 (Quadratic probing)
- Phương pháp băm kép (Double hashing)

Phương pháp dò tuyến tính

- **Phương pháp dò tuyến tính:**

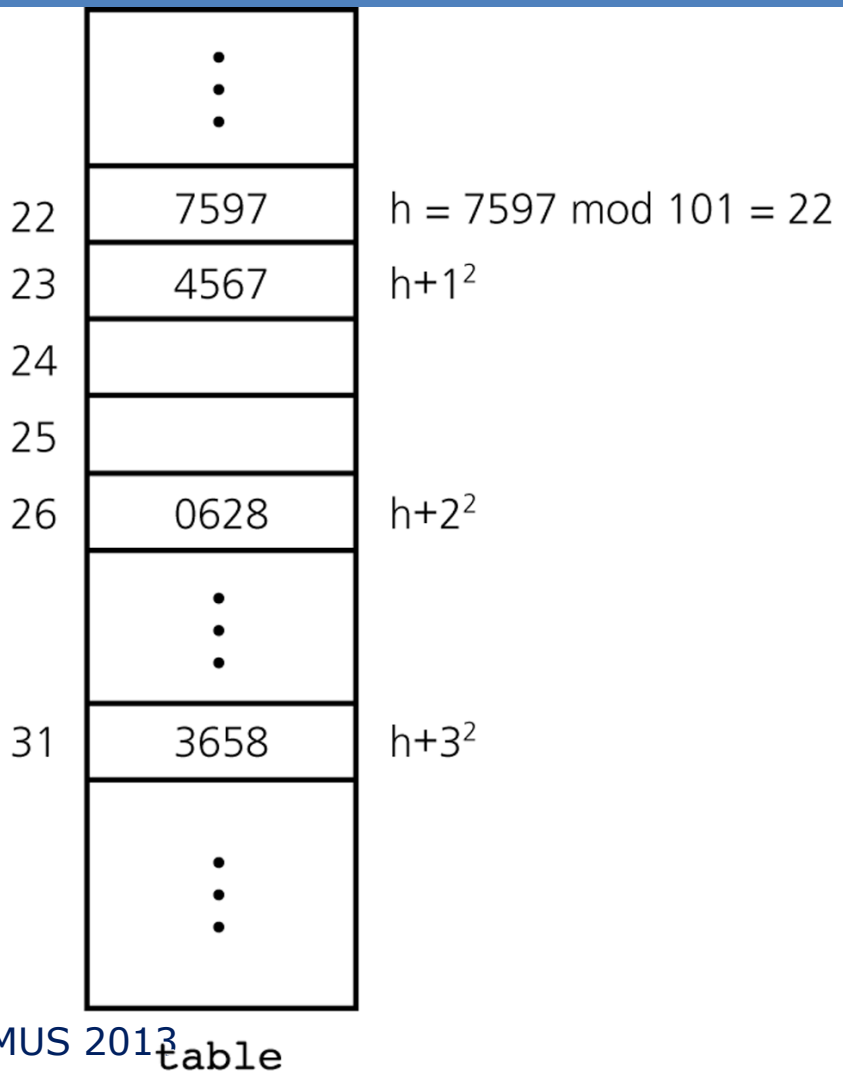
$H(k, i) = (h(k) + i) \bmod M$



Phương pháp dò bậc 2

- ***Phương pháp dò bậc 2:***

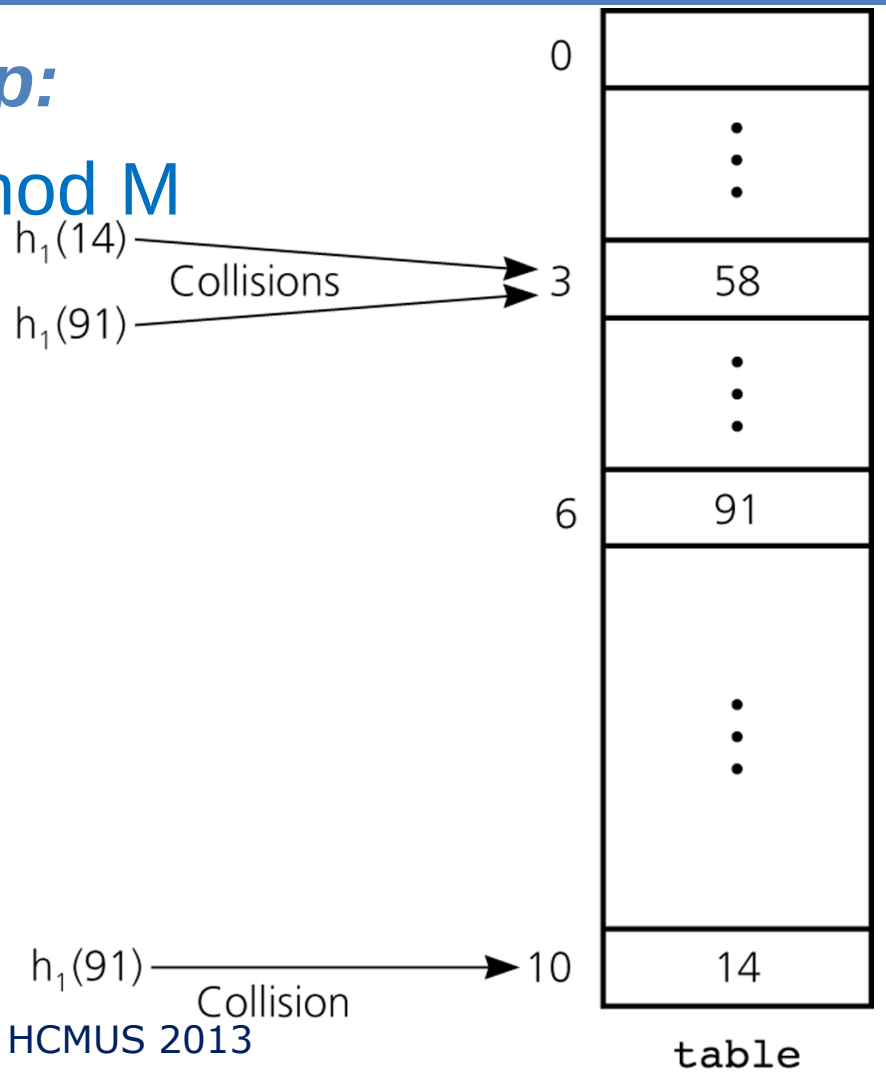
$H(k, i) = (h(k) + i^2) \bmod M$



Phương pháp băm kép

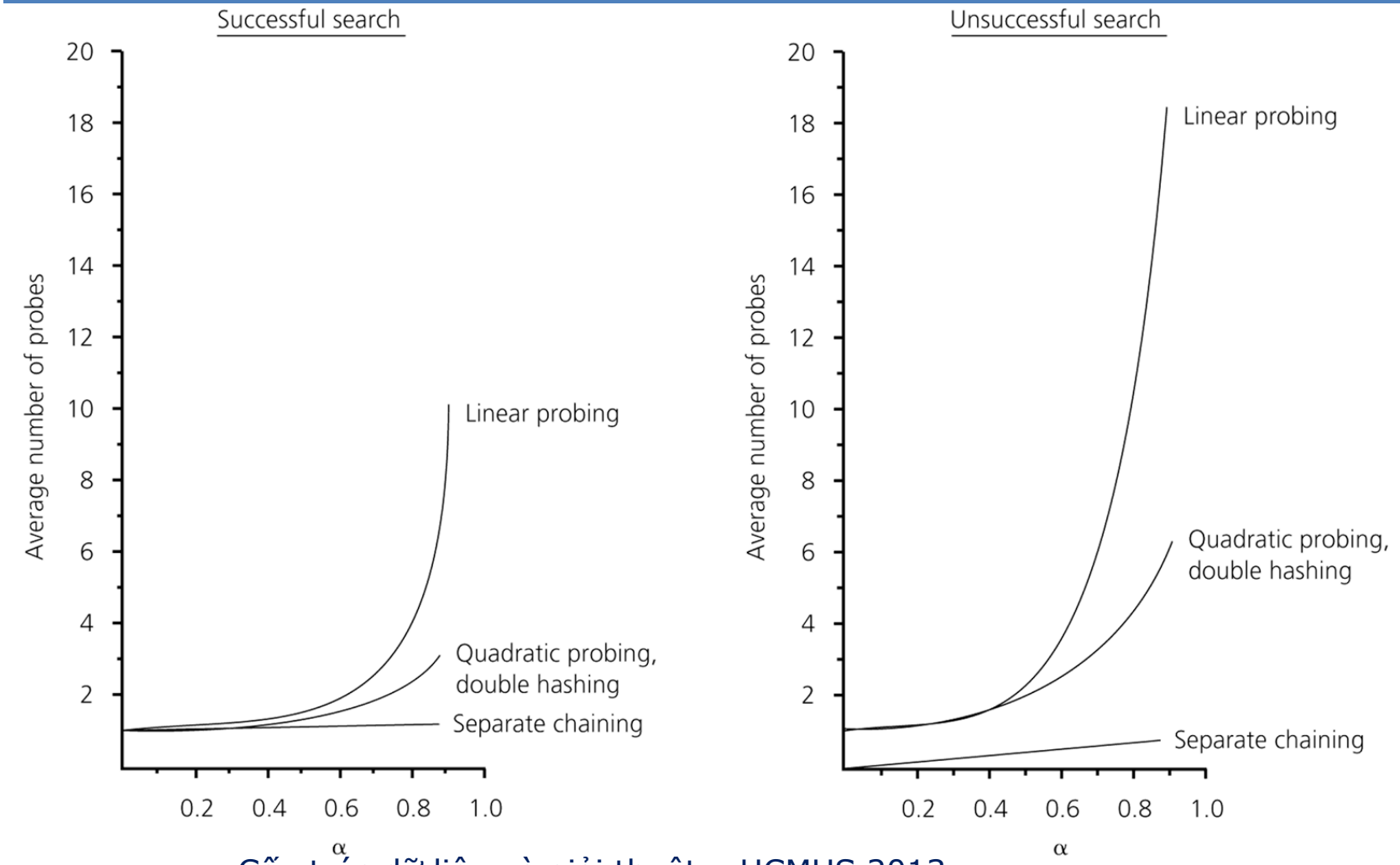
- ***Phương pháp băm kép:***

$H(k, i) = (h_1(k) + i*h_2(k)) \bmod M$



$$h_1(key) = key \bmod 11$$
$$h_2(key) = 7 - (key \bmod 7)$$

So sánh các phương pháp giải quyết đụng độ



Nhận xét

- Phương pháp địa chỉ mở:
  - Đơn giản khi cài đặt.
  - Sử dụng cấu trúc dữ liệu cơ bản.
  - Giải quyết được đụng độ nhưng lại có thể gây ra đụng độ mới.
- Phương pháp nối kết:
  - Không bị ảnh hưởng về tốc độ khi mảng gần đầy.
  - Ít tốn bộ nhớ khi mảng thưa (ít phần tử).

Bài tập

1. Cho bảng băm có kích thước M = 11. Hàm băm: h(k) = k mod M. Dùng phương pháp địa chỉ mở. Cho biết kết quả sau khi thêm vào bảng băm các khóa 10, 22, 31, 4, 15, 28, 17, 88, 59, với 3 phương pháp xử lý đụng độ:

   a. Dò tuyến tính.

   b. Dò bậc 2.

   c. Băm kép $h_2(k) = (k \bmod 19)+1$.

Bài tập

2. Cho từ điển Anh – Việt có 15.000 từ, hãy tổ chức cấu trúc dữ liệu bảng băm và cho biết hàm băm thích hợp giúp cho việc tra từ hiệu quả nhất.

Hỏi và Đáp