BẢNG BĂM (HASH TABLE)

DATA STRUCTURES AND ALGORITHMS

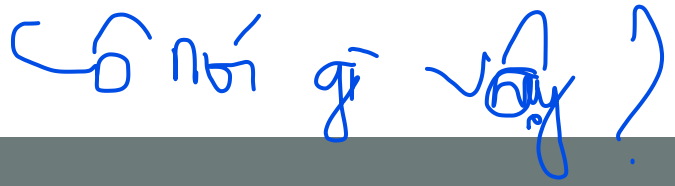
ThS Nguyễn Thị Ngọc Diễm
diemntn@uit.edu.vn

Nội dung

- **Các khái niệm trong Bảng băm**
- Các dạng Hàm băm – hash function
- Các phương pháp giải quyết đụng độ
  - Nối kết trực tiếp – Separate Chaining
  - Dò tuyến tính - Linear probing
  - Dò bậc hai - Quadratic probing
  - Băm kép - Double hashing
- Thư viện unodered_map, unodered_set

Giới thiệu

- Phép băm được đề xuất và hiện thực trên máy tính từ những năm 50 của thế kỷ 20. Dựa trên ý tưởng: chuyển đổi khóa thành một số (xử lý băm) và sử dụng số này để đánh chỉ số cho bảng dữ liệu.
- Các phép toán trên các cấu trúc dữ liệu như danh sách, cây nhị phân, … phần lớn được thực hiện bằng cách so sánh các phần tử của cấu trúc, do vậy thời gian truy xuất không nhanh và phụ thuộc vào kích thước của cấu trúc. Trong khi đó các phép toán trên bảng băm sẽ giúp hạn chế số lần so sánh, và vì vậy sẽ cố gắng giảm thiểu được thời gian truy xuất. Độ phức tạp của các phép toán trên bảng băm thường có bậc là O(1) và không phụ thuộc vào kích thước của bảng băm.



Các thuật ngữ thường dùng

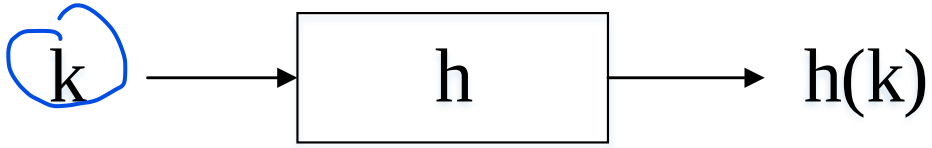
- Hash table (bảng băm)
- Hashing (phép băm)
- Hash function (hàm băm)
- Collision resolution (giải quyết đụng độ)
- Separate Chaining (Phương pháp nối kết)
- Open addressing (Địa chỉ mở)
- Load factor (Hệ số tải)

Các thuật ngữ thường dùng (tt)

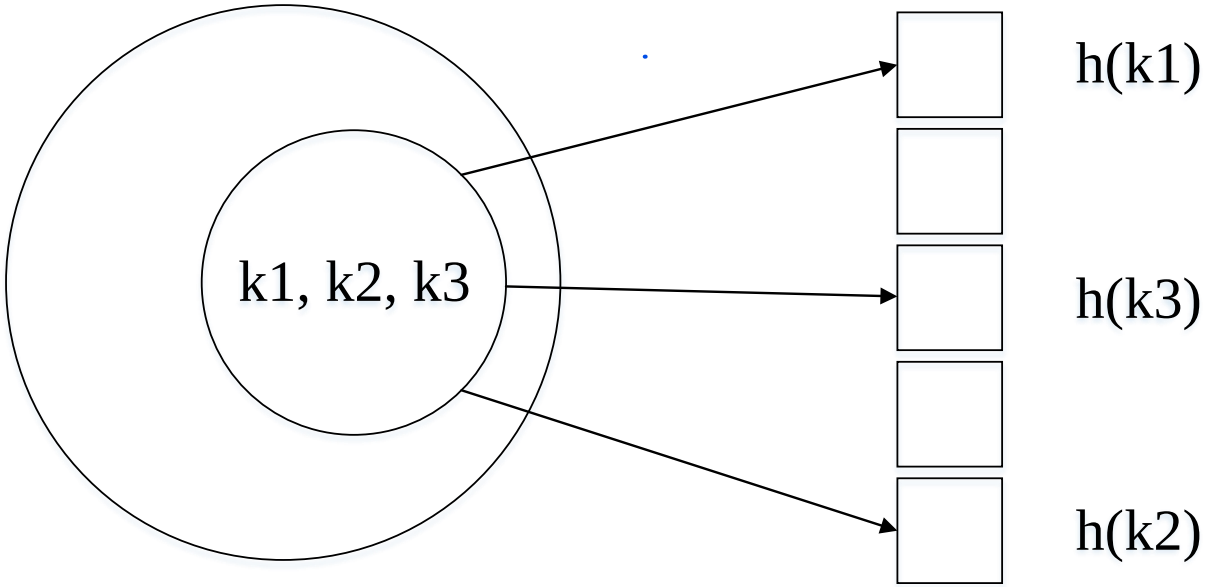
- Phép băm (Hashing): Là quá trình ánh xạ một giá trị khóa vào một vị trí trong bảng.
- Một hàm băm (Hash function) ánh xạ các giá trị khóa đến các vị trí ký hiệu: h hay HF
- Bảng băm (Hash Table) là mảng lưu trữ các record, ký hiệu: HT
- Hệ số tải (Load factor): là tỷ lệ giữa số phần tử trong bảng băm với kích thước bảng.
- HashTable có M vị trí được đánh chỉ mục từ 0 đến M-1, M là kích thước của bảng băm.
- Bảng băm thích hợp cho các ứng dụng được cài đặt trên đĩa và bộ nhớ, là một cấu trúc dung hòa giữa thời gian truy xuất và không gian lưu trữ.

Hàm băm (Hash Function)

- Hàm băm biến đổi một khóa vào một bảng các địa chỉ.



- Khóa có thể là dạng số hay số dạng chuỗi.
- Địa chỉ tính ra được là số nguyên trong khoảng 0 đến M-1 với M là số địa chỉ có trên bảng băm



Hàm băm (Hash Function)

- Hàm băm tốt thỏa mãn các điều kiện sau:
  - Tính toán nhanh
  - Các khoá được phân bố đều trong bảng
  - Ít xảy ra đụng độ
  - Xử lý được các loại khóa có kiểu dữ liệu khác nhau
- Giải quyết vấn đề băm với các khoá không phải là số nguyên:
  - Tìm cách biến đổi khóa thành số nguyên.
    - Ví dụ: loại bỏ dấu "-" trong "9635-8904" đưa về số nguyên 96358904
    - Đối với chuỗi, sử dụng giá trị các ký tự trong bảng mã ASCCI
- Sau đó sử dụng các hàm băm chuẩn trên số nguyên.

Load factor (Hệ số tải)

- Hệ số tải (Load factor): là tỷ lệ giữa số phần tử trong bảng băm với kích thước bảng.

$$\lambda = \frac{n}{m}$$

- Ví dụ: Hệ số tải của bảng băm T bên dưới là λ=5/8=0.63

0	1	2	3	4	5	6	7
a		**b**	**d**	**c**		**e**	

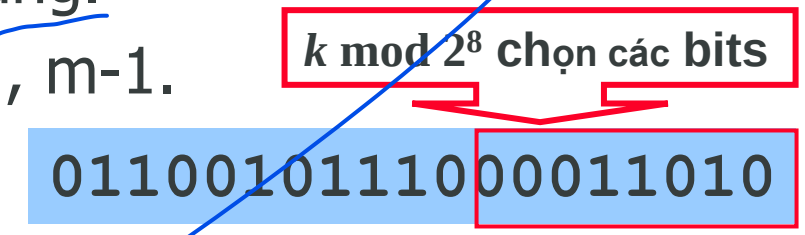
- Nếu hàm băm cho hệ số tải cụ thể, thì hàm băm chỉ được lưu trữ tối đa số lượng khóa tương ứng với hệ số tải.

Nội dung

- Các khái niệm trong Bảng băm
- **Các dạng Hàm băm – hash function**
- Các phương pháp giải quyết đụng độ
  - Nối kết trực tiếp – Separate Chaining
  - Dò tuyến tính - Linear probing
  - Dò bậc hai - Quadratic probing
  - Băm kép - Double hashing
- Thư viện unodered_map, unodered_set

HF: Phương pháp chia

- Dùng số dư: $h(k) = k \bmod m$
  - k là khoá, m là kích thước của bảng.
  - Như vậy h(k) sẽ nhận: 0, 1, 2, …, m-1.
- **Vấn đề chọn giá trị m**
  - $m = 2^n$: **không tốt** => giá trị của h(k) sẽ là n bits cuối cùng trong biểu diễn nhị phân của k.
  - $m = 10^n$: **không tốt** => giá trị của h(k) sẽ là n chữ số cuối cùng trong biểu diễn thập phân của k.
  - m là nguyên tố: **tốt**
    - Gia tăng sự phân bố đều
    - Thông thường m được chọn là số nguyên tố gần với 2^n
    - Chẳng hạn bảng ~4000 mục, chọn $m = 4093$



Ví dụ

- Đề bài: Cho bảng băm T có kích thước m=11, khóa là mã hàng gồm 3 ký tự ASCII. Hãy xây dựng hàm băm dùng phép chia cho T và tính chỉ số cho các mã hàng "ATK", "SOS" và "TNT".
- Bài giải:
  - Biến đổi chuỗi ký tự thành số:

    toInt(s) = s[0] + s[1] * 37 + s[2] * 372
  - Hàm băm: h(s) = toInt(s) mod 11
  - Khi đó:

    h("ATK") = (65 + 84 * 37 + 75 * 372) mod 11 = 6

    h("SOS") = (83 + 79 * 37 + 83 * 372) mod 11 = 0

    h("TNT") = (84 + 78 * 37 + 84 * 372) mod 11 = 2

Ví dụ: Một số cách chuyển chuỗi thành số

- Xem một số cách chuyển chuỗi thành số trình bày tại mục 5.2: Hash function trong tài liệu: "Mark Allen Weiss, Data Structures and Algorithm Analysis in C++, 2004"

```
1   int hash( const string & key, int tableSize )
2   {
3       int hashVal = 0;
4
5       for( char ch : key )
6           hashVal += ch;
7
8       return hashVal % tableSize;
9   }
```

Figure 5.2 A simple hash function

```
1   /**
2    * A hash routine for string objects.
3    */
4   unsigned int hash( const string & key, int tableSize )
5   {
6       unsigned int hashVal = 0;
7
8       for( char ch : key )
9           hashVal = 37 * hashVal + ch;
10
11      return hashVal % tableSize;
12  }
```

Figure 5.4 A good hash function

```
1   int hash( const string & key, int tableSize )
2   {
3       return ( key[ 0 ] + 27 * key[ 1 ] + 729 * key[ 2 ] ) % tableSize;
4   }
```

Figure 5.3 Another possible hash function—not too good

HF: Phương pháp nhân

- Sử dụng: **h(k) = floor(m(kA mod 1))**
  - k là khóa, m là kích thước bảng, A là hằng số với 0 < A < 1 và (kA mod 1) là phần thập phân kA.
- Chọn m và A
  - Lợi thế của phương pháp nhân là nó hoạt động tốt như nhau với bất kỳ kích thước m. Và thường thì người ta chọn lũy thừa của 2: $m = 2^p$
  - Sự tối ưu trong việc chọn A phụ thuộc vào đặc trưng của dữ liệu. Theo Knuth chọn $A = ½(\sqrt{5} - 1) \approx 0.6180339887$ được xem là tốt.
- Ví dụ:
  - k = 270589; m = 1000
  - h(k) = 1000 * (270589* 0.6180339887 mod 1)
  - h(k) = 1000 * (167233.1990 mod 1)
  - h(k) = 1000* 0.199 = 199

Hàm băm phổ quát

- Một họ các hàm băm ***H*** được gọi là phổ quát nếu với hai khóa x ≠ y bất kỳ thì số lượng hàm băm $h \in \boldsymbol{H}$ có h(x) = h(y) không vượt quá $|\boldsymbol{H}|/m$.
- Nếu hàm băm $h \in \boldsymbol{H}$ thì xác suất xảy ra đụng độ khi sử dụng hàm băm h không vượt quá 1/m.
- ***Định lý***:
  - Họ hàm băm: $\boldsymbol{H=\{ H_{a,b}(k)=((a*k+b) \bmod p) \bmod m \}}$
  - Trong đó:
    - a, b là hai số tự nhiên tùy ý, $1 \le a \le p-1$, $0 \le b \le p-1$
    - p là số nguyên tố lớn hơn tất cả các giá trị khóa.

Phép băm phổ quát

- Ví dụ: Xác định hàm băm cho bảng băm T có kích thước 11 và có giá trị khóa lớn nhất là 48.

- Chọn p = 53 là số nguyên tố lớn hơn 48.

- Chọn a = 2 và b = 1.

- Hàm băm cần tìm là:

  - h(k) = ((2 * k + 1) mod 53) mod 11

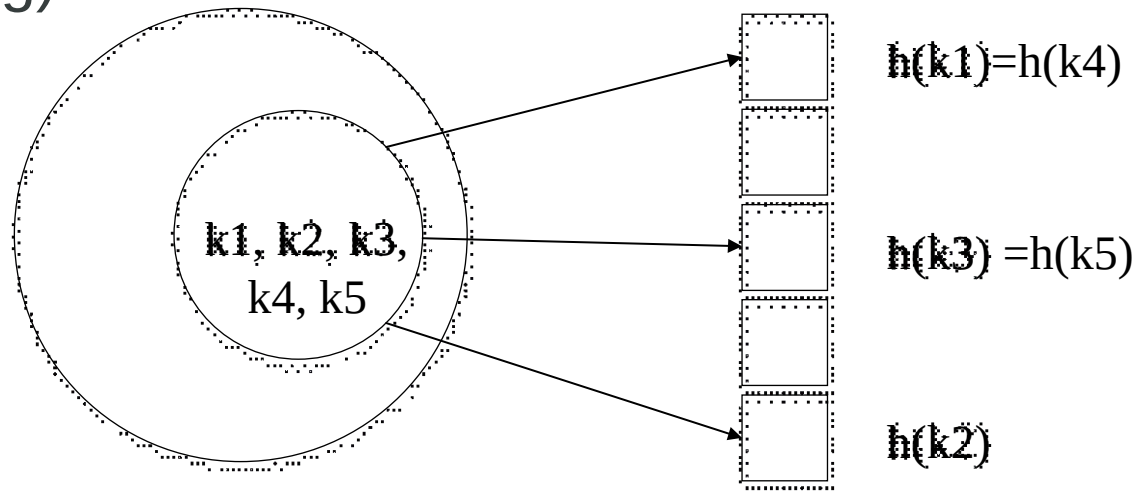
Nội dung

- Các khái niệm trong Bảng băm
- Các dạng Hàm băm – hash function
- **Các phương pháp giải quyết đụng độ**
  - Nối kết trực tiếp – Separate Chaining
  - Dò tuyến tính - Linear probing
  - Dò bậc hai - Quadratic probing
  - Băm kép - Double hashing
- Thư viện unodered_map, unodered_set

Sự đụng độ (Collision)

- Sự đụng độ là hiện tượng các khóa khác nhau nhưng băm cùng địa chỉ như nhau.
- Khi key1<>key2 mà h(key1)=h(key2) chúng ta nói nút có khóa key1 đụng độ với nút có khóa key2.
- Thực tế người ta giải quyết sự đụng độ theo hai phương pháp: phương pháp nối kết (chaining) và phương pháp băm lại (~~open adressing~~).

open addressing



Quy trình thực hiện lưu trữ bằng bảng băm

Quy trình thực hiện lưu trữ bằng bảng băm được thực hiện qua 2 bước:

- *Bước 1:* Xác định hàm băm để biến đổi khóa cần tìm thành địa chỉ trong bảng băm.
- *Bước 2:* Giải quyết đụng độ (collision) cho trường hợp những khóa khác cho ra cùng một địa chỉ trong bảng băm.

Giải quyết xung đột - Collision Resolution

Gồm các phương pháp:

1. Các loại bảng băm giải quyết sự xung đột bằng phương pháp nối kết (Separate Chaining) như:
   - Nối kết trực tiếp - Direct Chaining
   - Nối kết hợp nhất - Coalesced Chaining (chỉ giới thiệu)

- Các phần tử bị băm cùng địa chỉ (các phần tử bị xung đột) được gom thành một danh sách liên kết. Lúc này mỗi phần tử trên bảng băm cần khai báo thêm trường liên kết next chỉ phần tử kế bị xung đột cùng địa chỉ.
- Bảng băm giải quyết sự xung đột bằng phương pháp này cho phép tổ chức các phần tử trên bảng băm rất linh hoạt: khi thêm một phần tử vào bảng băm chúng ta sẽ thêm phần tử này vào danh sách liên kết thích hợp phụ thuộc vào băm. Tuy nhiên bảng bảng băm loại này bị hạn chế về tốc độ truy xuất.

Giải quyết xung đột - collision Resolution

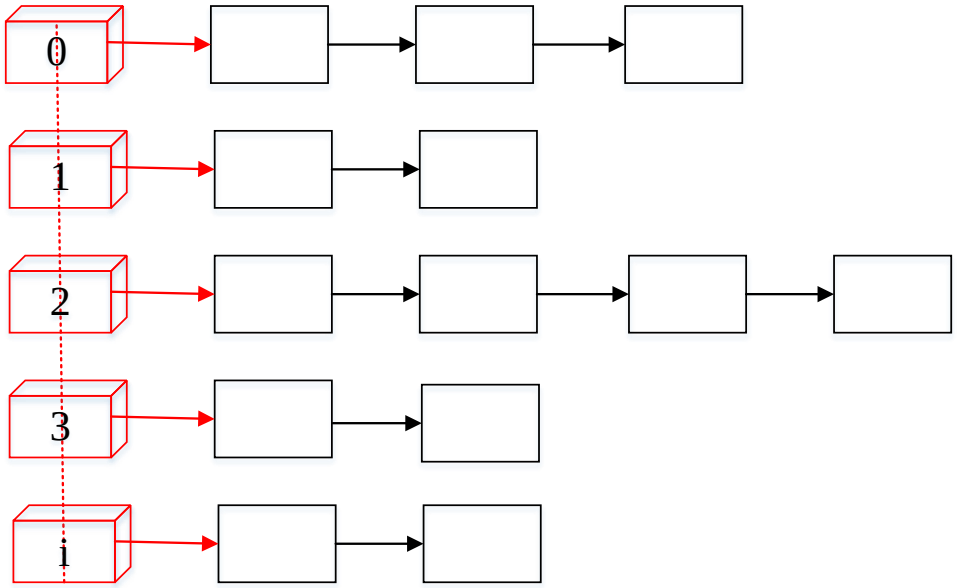
2. Các loại bảng băm giải quyết sự xung đột bằng phương pháp băm lại như: (open addressing)

- Dò tuyến tính - Linear probing
- Dò bậc hai - Quadratic probing
- Băm kép - Double hashing

- Nếu băm lần đầu bị xung đột thì băm lại lần 1, nếu bị xung đột nữa thì băm lai lần 2,… Quá trình băm lại diễn ra cho đến khi không còn xung đột nữa. Các phép băm lại (rehash function) thường sẽ chọn địa chỉ khác cho các phần tử.
- Để tăng tốc độ truy xuất, các bảng băm giải quyết sự xung đột bằng phương pháp băm lại thường được cài đặt bằng danh sách kề. Tuy nhiên việc tổ chức các phần tử trên bảng băm không linh hoạt vì các phần tử chỉ được lưu trữ trên một danh sách kề có kích thước đã xác định trước.

Direct Chaining Method - PP nối kết trực tiếp

- Các nút bị băm cùng địa chỉ (các nút bị xung đột) được gom thành một danh sách liên kết.
- Các nút trên bảng băm được *băm* thành các danh sách liên kết. Các nút bị xung đột tại địa chỉ i được nối kết trực tiếp với nhau qua danh sách liên kết i.



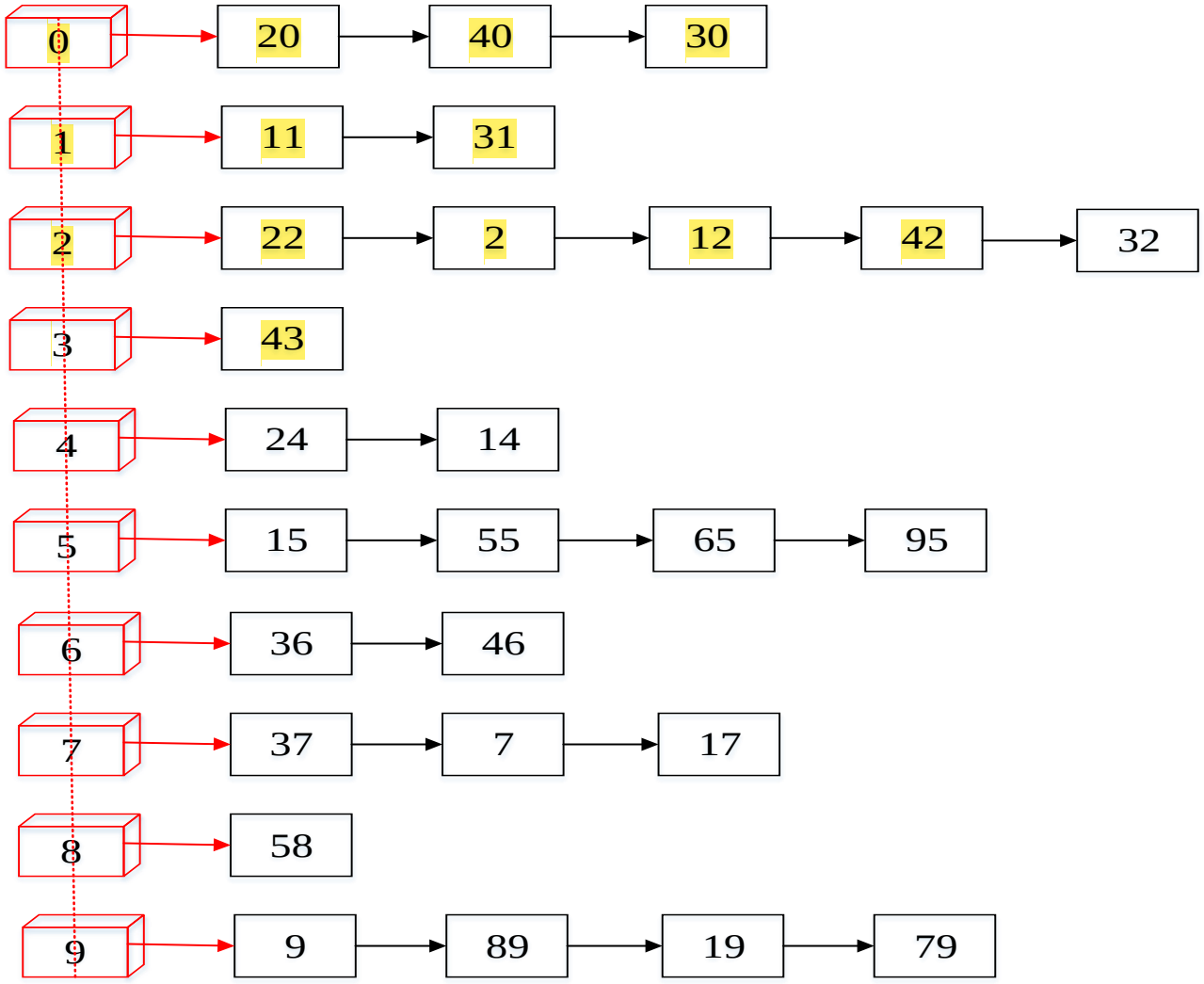
Direct Chaining Method - PP nối kết trực tiếp

- *Khi thêm một phần tử* có khóa k vào bảng băm, hàm băm h(k) sẽ xác định địa chỉ i trong khoảng từ 0 đến M-1 ứng với danh sách liên kết i mà phần tử này sẽ được thêm vào.
- *Khi tìm một phần tử* có khóa k vào bảng băm, hàm băm h(k) cũng sẽ xác định địa chỉ i trong khoảng từ 0 đến M-1 ứng với danh sách liên kết i có thể chứa phần tử này. Như vậy, việc tìm kiếm phần tử trên bảng băm sẽ được qui về bài toán tìm kiếm một phần tử trên danh sách liên kết.

Direct Chaining Method: Ví dụ minh họa

- Xét bảng băm có cấu trúc như sau:
  - Tập khóa K: tập số tự nhiên
  - Tập địa chỉ M: gồm 10 địa chỉ (M={0, 1, …, 9}
  - Hàm băm h(key) = key % 10.
- Hình trên minh họa bảng băm vừa mô tả. Theo hình vẽ, bảng băm đã "băm" phần tử trong tập khoá K theo 10 danh sách liên kết khác nhau, mỗi danh sách liên kết gọi là một bucket:
  - Bucket 0 gồm những phần tử có khóa tận cùng bằng 0.
  - Bucket i(i=0 | … | 9) gồm những phần tử có khóa tận cùng bằng i. Để giúp việc truy xuất bảng băm dễ dàng, các phần tử trên các bucket cần thiết được tổ chức theo một thứ tự, chẳng hạn từ nhỏ đến lớn theo khóa.
  - Khi khởi động bảng băm, con trỏ đầu của các bucket là NULL.
- Theo cấu trúc này, với tác vụ insert, hàm băm sẽ được dùng để tính địa chỉ của khoá k của phần tử cần chèn, tức là xác định được bucket chứa phần tử và đặt phần tử cần chèn vào bucket này.
- Với tác vụ search, hàm băm sẽ được dùng để tính địa chỉ và tìm phần tử trên bucket tương ứng.

Direct Chaining Method: Ví dụ minh họa



(Chú ý: Khi làm bài thi cần trình bày rõ vì sao thêm khóa k vào vị trí nào i trong bảng băm)

Direct Chaining Method: Nhận xét

- Bảng băm dùng phương pháp nối kết trực tiếp sẽ ”băm” n nút vào M danh sách liên kết (M buckets).
- Tốc độ truy xuất phụ thuộc vào việc lựa chọn hàm băm sao cho băm đều n nút của bảng băm cho M buckets.
- Nếu chọn M càng lớn thì tốc độ thực hiện các phép toán trên bảng băm càng nhanh tuy nhiên không hiệu quả về bộ nhớ. Chúng ta có thể điều chỉnh M để dung hòa giữa tốc độ truy xuất và dung lượng bộ nhớ.
- Nếu chọn M=n thời gian truy xuất tương đương với truy xất trên mảng (có bậc O(1)), song tốn bộ nhớ.
- Nếu chọn M = n /k (k =2, 3, 4, …) thì ít tốn bộ nhớ hơn k lần, nhưng tốc độ chậm đi k lần.

Direct Chaining Method: Nhận xét (tt)

	Worst Case			Average Case		
Implementation	Search	Insert	Delete	Search	Insert	Delete
Sorted array	logN	N	N	logN	N/2	N/2
Unsorted array	N	N	N	N/2	N	N/2
Direct Chaining	N	N	N	1*	1*	1*

*assumes hash function is random

Direct Chaining Method: Cài đặt

Cài đặt HashTable sử dụng phương pháp giải quyết đụng độ Sepreate Chainging lưu trữ số nguyên với Hằm băm h(key) = key % M

```
struct NODE {
    int key;
    NODE *pNext;
};
struct LIST {
    NODE * pHead, *pTail;
};
/* Khai báo cấu trúc HASHTABLE */
struct HASHTABLE {
    int M; // Kích thước bảng băm
    int n; // Số phần tử hiện tại trong bảng băm
    LIST *table;
};
```

Direct Chaining Method: Cài đặt

Hàm băm: Giả sử chúng ta chọn hàm băm h(key) = key % M.

```
int HF(HASHTABLE ht, int key) { return key % ht.M; }
```

Phép toán khởi tạo bảng băm rỗng: Khởi động các HASHTABLE.

```
void CreateEmptyHashTable(HASHTABLE &H){
    cin >> H.M;
    H.table = new LIST[H.M];
    for (int i = 0; i < H.M; i++)
        CreateEmptyList(H.table[i]);
}
```

Direct Chaining Method: Cài đặt (tt)

- **Phép toán Insert:**
  - Bước 1: Xác định vị trí DSLK phù hợp trên HASHTABLE để chèn key vào bảng băm
  - Dùng phép toán `InsertToLinkedList` (có thể là AddTail, AddHead, …) để thêm phần tử key vào DSLK tìm được ở Bước 1.

```
void Insert(HASHTABLE &H, int key) {
// Kiểm tra điều kiện Load Factor nếu có yêu cầu
// VD: if(H.n/H.M>LOAD)  return 0;
    int i = HF(H, key);
    LIST L=H[i];
    InsertToLinkedList(L, key);
}
```

Direct Chaining Method: Cài đặt (tt)

- **Phép toán loại bỏ:**
  - Bước 1: Xác định vị trí DSLK đơn chứa key trên HASHTABLE
  - Bước 2: Dùng phép toán `Search_X_In_LinkedList` để tìm giá trị key trong DSLK tìm được ở Bước 1.

```
void Delete(HASHTABLE &H, int key) {
    int i = HF(H, key);
    LIST L=H[i];
    Search_X_In_LinkedList(L, key);
}
```

Direct Chaining Method: Cài đặt (tt)

- **Phép toán loại bỏ:**
  - Bước 1: Xác định vị trí DSLK đơn chứa key trên HASHTABLE
  - Bước 2: Dùng phép toán `Delete_X_In_LinkedList` để xóa phần tử key khỏi DSLK tìm được ở Bước 1.

```
void Delete(HASHTABLE &H, int key) {
    int i = HF(H, key);
    LIST L=H[i];
    Delete_X_In_LinkedList(L, key);
}
```

Direct Chaining Method: Cài đặt (tt)

Phép toán duyệt HASHTABLE[i]:

Duyệt các phần tử trong HASHTABLE thứ i:

```
void traverseHASHTABLE(HASHTABLE H, int i) {
    NODE* p = H[i].pHead;
    while (p != NULL) {
        cout << p->key << "\t";
        p = p->pNext;
    }
}
```

Direct Chaining Method: Cài đặt (tt)

Phép toán duyệt toàn bộ bảng băm:

Duyệt toàn bộ bảng băm.

```
void traverse(HASHTABLE H) {
   for (int i = 0; i<H.M; i++) {
      cout << endl << "Butket " << i << ": ";
      traverseHASHTABLE(H, i);
   }
}
```

Coalesced Chaining Method - PP nối kết hợp nhất

- (Chỉ giới thiệu, không thi)

key	next
nullkey	-1
...	...
nullkey	-1

Coalesced Chaining

- Bảng băm trong trường hợp này được cài đặt bằng danh sách liên kết dùng mảng, có M nút. Các nút bị xung đột địa chỉ được nối kết nhau qua một danh sách liên kết.
- Mỗi nút của bảng băm là một mẫu tin có 2 trường:
  - Trường key: chứa các khóa node
  - Trường next: con trỏ chỉ node kế tiếp nếu có xung đột.
- Khi khởi động bảng băm thì tất cả trường key được gán nullkey, tất cả trường next được gán -1.

Coalesced Chaining

- Khi thêm một nút có khóa key vào bảng băm, hàm băm H(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến M-1.
- Nếu chưa bị xung đột thì thêm nút mới vào địa chỉ này .
- Nếu bị xung đột thì nút mới được cấp phát là nút trống phía cuối mảng. Cập nhật liên kết next sao cho các nút bị xung đột hình thành một danh sách liên kết.
- Khi tìm một nút có khóa key trong bảng băm, hàm băm H(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến M-1, tìm nút khóa key trong danh sách liên kết xuất phát từ địa chỉ i.

Coalesced Chaining Method: Ví dụ 1

- Minh họa cho bảng băm có tập khóa là tập số tự nhiên, tập địa chỉ có 10 địa chỉ (M=10) (từ địa chỉ 0 đến 9), chọn hàm băm h(key) = key mod 10.
- Tập keys = {30, 24, 26, 10, 14, 54, 4}

(Chú ý: Khi làm bài thi cần trình bày rõ vì sao thêm khóa k vào vị trí nào i trong bảng băm)

0	30	9
1		-1
2		-1
3		-1
4	24	8
5	4	-1
6	26	-1
7	54	5
8	14	7
9	10	-1

Coalesced Chaining Method: Ví dụ 2

- Minh họa cho bảng băm có tập khóa là tập số tự nhiên, tập địa chỉ có 10 địa chỉ (M=10) (từ địa chỉ 0 đến 9), chọn hàm băm h(key) = key mod 10.
- Tập keys = {30, 24, 26, 10, 14, 54, 4, 8, 84}

(Chú ý: Khi làm bài thi cần trình bày rõ vì sao thêm khóa k vào vị trí nào i trong bảng băm)

0	30	9
1		-1
2	84	-1
3	8	2
4	24	8
5	4	3
6	26	-1
7	54	5
8	14	7
9	10	-1

Coalesced Chaining Method: Ví dụ 3

- Minh họa cho bảng băm có tập khóa là tập số tự nhiên, tập địa chỉ có 10 địa chỉ (M=10) (từ địa chỉ 0 đến 9), chọn hàm băm h(key) = key mod 10.
- Tập keys = {20, 31, 10, 51, 84, 50, 1, 24, 90}

(Chú ý: Khi làm bài thi cần trình bày rõ vì sao thêm khóa k vào vị trí nào i trong bảng băm)

0	20	9
1	31	8
2	nullkey	-1
3	90	-1
4	84	5
5	24	-1
6	1	-1
7	50	3
8	51	6
9	10	7

Coalesced Chaining Method: Nhận xét

- Thực chất cấu trúc bảng băm này chỉ tối ưu khi băm đều, nghĩa là mỗi danh sách liên kết chứa một vài phần tử bị xung đột, tốc độ truy xuất lúc này có bậc O(1).
- Trường hợp xấu nhất là băm không đều vì hình thành một danh sách có n phần tử nên tốc độ truy xuất lúc này có bậc O(n).

Coalesced Chaining Method: Cài đặt

Chương trình cài đặt bằng danh sách kề

- **Khai báo cấu trúc bảng băm:**

```
#define nullkey -1

#define M 100

// Khai báo cấu trúc một node trong bảng băm

struct NODE{
    int key;
    int next; // con trỏ chỉ nút kế tiếp khi có xung đột

};

// Khai báo bảng băm

NODE HASHTABLE[M];

int avail; // biến toàn cục chỉ nút trống ở cuối bảng
băm được cập nhật khi có xung đột
```

Coalesced Chaining Method: Cài đặt (tt)

- **Hàm băm:**

```
// Giả sử chúng ta chọn hàm băm dạng mod: h(key) = key % 10.
Có thể dùng một hàm băm bất kì thay cho hàm băm dạng % trên.

int HF(int key) {
    return key % 10;

}
```

- **Phép toán khởi tạo:**

```
void Initialize() {
    for (int i = 0; i<M; i++){
        HASHTABLE[i].key = nullkey; HASHTABLE[i].next = -1;
    }
    avail = M - 1; // nút M-1 là nút ở cuối bảng chuẩn bị cấp
    phát nếu có xung đột

}
```

Coalesced Chaining Method: Cài đặt (tt)

- **Phép toán tìm kiếm:**

```
int Search(int k) {
    int i = HF(k);
    while (k != HASHTABLE[i].key && i != -1)
          i = HASHTABLE[i].next;

    if (k == HASHTABLE[i].key)
          return i; //tìm thấy

    return M; //không tìm thấy
}
```

Coalesced Chaining Method: Cài đặt (tt)

- **Phép toán lấy phần tử trống cuối bảng băm:**

Chọn phần tử còn trống phía cuối bảng băm để cấp phát khi xảy ra xung đột.

```
int getEmpty() {
   while (HASHTABLE[avail].key != nullkey)
         avail --;
   return avail;
}
```

Coalesced Chaining Method: Cài đặt (tt)

- **Phép toán Insert**

```
int Insert(int k) {
    int i = Search(k), j; // j là địa chỉ nút trong được cấp phát
    if (i != M) {
      cout << "\n Khoa " << k << " bi trung, khong them nut nay duoc!";
      return i;
    }
    i = HF(k);
    while (HASHTABLE[i].next >= 0) i = HASHTABLE[i].next;
    if (HASHTABLE[i].key == nullkey) j = i; // Nút i còn trống thì cập nhật
    else { // Nếu nút i là nút cuối của danh sách
       j = getEmpty();
       if (j < 0) { cout << "\n Bang bam bi day!“; return j; }
       HASHTABLE[i].next = j;
    }
    HASHTABLE[j].key = k;
    return j;
}
```

Open addressing

H(key, i) = (h(key) + f(i)) % TableSize

Trong đó:

- *key: từ khóa cần băm*
- *h(key): hàm băm chính.*
- *H(key, i): hàm băm lại lần thứ i.*
- *TableSize: kích thước bảng băm.*
- *%: phép lấy dư (mod)*
- *f(i): hàm thể hiện của các phương pháp xử lý đụng độ: dò tuyến tính, dò bậc hai, băm kép, f(0)=0*

* Phương pháp này không sử dụng con trỏ

Open addressing (tt)

$$H(key, i) = (h(key) + f(i)) \% TableSize$$

- Với phương pháp dò tuyến tính thì f(i) = i

$$H(key, i) = (h(key) + i) \% TableSize$$

- Với phương pháp dò bậc 2 thì $f(i) = i^2$

$$H(key, i) = (h(key) + i^2) \% TableSize$$

- Với phương pháp băm kép thì f(i) = i * h2(key)

$$H(key, i) = (h1(key) + i * h2(key)) \% TableSize$$

Linear Probing Method - PP dò tuyến tính

- Hàm băm lại của phương pháp dò tuyến tính là truy xuất địa chỉ kế tiếp. Hàm băm lại lần i được biểu diễn bằng công thức sau:

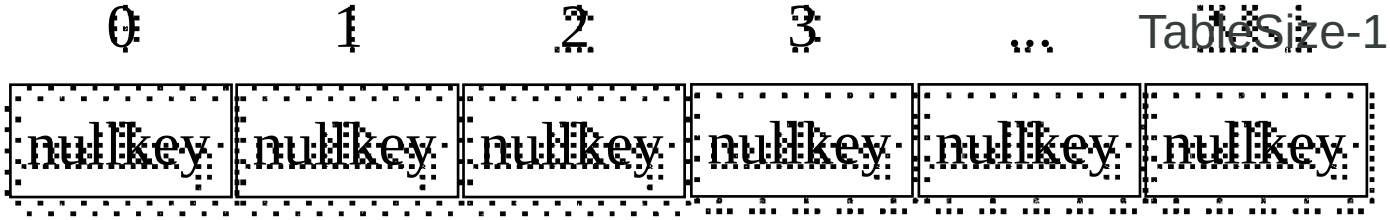
$$H(key, i) = (h(key) + i) \bmod TableSize$$

với h(key) là hàm băm chính của bảng băm, f(i)=i

- Lưu ý địa chỉ dò tìm kế tiếp là địa chỉ 0 nếu đã dò đến cuối bảng.

Linear Probing Method

- Bảng băm trong trường hợp này được cài đặt bằng danh sách kề có TableSize nút, mỗi nút của bảng băm là một mẫu tin có một trường key để chứa khóa của nút. So at any point, size of the table must be greater than or equal to the total number of keys
- Khi khởi động bảng băm thì tất cả trường key được gán nullkey.
- Khi thêm nút có khoá key vào bảng băm, hàm băm h(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến TableSize-1:
  - Nếu chưa bị xung đột thì thêm phần tử mới vào địa chỉ này.
  - Nếu bị xung đột thì hàm băm lại lần 1, hàm h_1 sẽ xét địa chỉ kế tiếp, nếu lại bị xung đột thì hàm băm thì hàm băm lại lần 2, hàm h_2 sẽ xét địa chỉ kế tiếp nữa, …, và quá trình cứ thế cho đến khi nào tìm được địa chỉ trống và thêm phần tử mới vào địa chỉ này.



Linear Probing Method

- Nếu chưa bị xung đột thì thêm nút mới vào địa chỉ này.
- Nếu bị xung đột thì hàm băm lại lần 1 (H(key, 1)) sẽ xét địa chỉ kế tiếp, nếu lại bị xung đột thì hàm băm lại lần 2 (H(key, 2)) sẽ xét địa chỉ kế tiếp …
- Quá trình cứ thế cho đến khi nào tìm được địa chỉ trống và thêm nút vào địa chỉ này.
- Khi tìm một nút có khoá key vào bảng băm, hàm băm h(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến TableSize-1, tìm nút khoá key trong khối đặc chứa các nút xuất phát từ địa chỉ i.

Linear Probing Method (dò tuyến tính): Ví dụ

- Giả sử, khảo sát bảng băm có cấu trúc như sau:
  - Tập khóa K: tập số tự nhiên
  - Tập địa chỉ TableSize: gồm 10 địa chỉ (TableSize ={0, 1, …, 9})
  - Hàm băm h(key) = key mod 10.
- Hình thể hiện thêm các giá trị 41, 45, 51, 30, 62, 80, 53, 89, 77, 19 vào bảng băm.

- Hàm băm dò tuyến tính:

$$H(key, i) = (\ key \bmod 10 + i) \% 10$$

Linear Probing Method: Ví dụ (tt)

- Thêm: {41, 45, 51, 30, 62, 80, 53, 89, 77, 19},

h(key) = key mod 10, H(key, i) = (h(key) + i) % 10

Bài giải:

- h(51)=51 mod 10=1 => đụng độ

➔ Băm lại: H(51, 1) = (1 + 1) % 10=2

- h(30)=30 mod 10=0
- h(62)=2 => đụng độ

➔ Băm lại: H(62, 1) = (2 + 1) % 10=3

- h(80)=80 mod 10=0 => đụng độ

➔ Băm lại: H(80, 1) = (0 + 1) % 10=1 => đụng độ

➔ Băm lại: H(80, 2) = (0 + 2) % 10=2 => đụng độ

➔ Băm lại: H(80, 3) = (0 + 3) % 10=3 => đụng độ

➔ Băm lại: H(80, 4) = (0 + 4) % 10=4

Tương tự chèn 53, 89, 77, 19

	Kết quả
0	30
1	41
2	51
3	62
4	80
5	45
6	53
7	77
8	
9	89

Linear Probing Method: Ví dụ (tt)

Quá trình thay đổi HashTable:

	Empty Table	Thêm 41	Thêm 45	Thêm 51	Thêm 30	Thêm 62	Thêm 80	Thêm 53	Thêm 89	Thêm 77	Thêm 19
0					30	30	30	30	30	30	30
1		41	41	41	41	41	41	41	41	41	41
2				51	51	51	51	51	51	51	51
3						62	62	62	62	62	62
4							80	80	80	80	80
5			45	45	45	45	45	45	45	45	45
6								53	53	53	53
7										77	77
8											19
9									89	89	89

Chèn trong Linear Probing HB with Load factor

- Thêm: {41, 45, 51, 30, 62, 80, 53, 89, 77, 19}

h(key) = key mod 10 load factor: lamda = n/m

Hàm băm lại: H(key, i) = (h(key) + i) % 10

Hệ số tải của bảng băm được ấn định là 0.5, nghĩa là bảng băm luôn đảm bảo số phần tử được lưu trong bảng băm không quá 50% kích thước của bảng băm.

Bài giải:

Lưu ý trong quá trình băm phải kiểm tra hệ số tải trước khi thêm key vào HashTable.

- h(41)=41, h(45)=45
- h(51)=51 mod 10=1 => đụng độ

➔ Băm lại: H(51, 1) = (1 + 1) % 10=2

- h(30)=30 mod 10=0
- h(62)=2 => đụng độ

➔ Băm lại: H(62, 1) = (2 + 1) % 10=3

	Kết quả
0	30
1	41
2	51
3	62
4	
5	45
6	
7	
8	
9	

Tìm kiếm trong Linear Probing HashTable

- Tìm 30:

h(30)=0 => Tìm thấy 30 tại vị trí 0

- Tìm 51:

h(51)=1 => Đụng độ

➔ Băm lại: H(51, 1)=2 => Tìm thấy 51

- Tìm 61:

h(61)=1 => Đụng độ

➔ Băm lại H(61, 1)=2 => Đụng độ

➔ Băm lại H(61, 2)=3 => Đụng độ

➔ Băm lại H(61, 3)=4 => Vị trí trống => KHÔNG TÌM THẤY

```
Search(HashTable H, int x)
{
while(true)
{
if(H[h(x)] == x) return tim thay, dung
if(H[h(x)] == empty return dung
if(H[h(x)] == deleted or occupied = bam lai
}
}
```

	Kết quả
0	30
1	41
2	51
3	62
4	
5	45
6	
7	
8	
9	

Xóa trong Linear Probing HashTable

- Xóa 41:

h(41)=0 => Tìm thấy 41 tại vị trí 1

⇒Xóa 41

- Tìm 51:

h(51)=1 => Vị trí đã bị xóa => Cách xử lý?

```
Search(HashTable H, int x)
{
    while(true)
    {
        if(H[h(x)] == x) return tim thay, dung
        if(H[h(x)] == empty return dung
        if(H[h(x)] == deleted or occupied = bam lai f(i)
    }
}
```

	Kết quả
0	30
1	~~41~~
2	51
3	62
4	
5	45
6	
7	
8	
9	

Xóa trong Linear Probing HashTable

- Sử dụng ba trạng thái khác nhau của một vị trí:
  - **Có giá trị (OCCUPIED)**
  - **Đã xóa (trước đây có giá trị nhưng đã bị xóa) (DELETED)**
  - **Trống (EMPTY)**
- Khi một giá trị bị xóa khỏi bảng băm, chuyển trạng thái của vị trí là “**DELETED**", thay vì làm trống vị trí.

Xóa trong Linear Probing HashTable

- Xóa 41:

h(41)=0 => Tìm thấy 41 tại vị trí 1

➔ Xóa 41: Chuyển trạng thái vị trí chứa 41 thành DELETED

- Tìm 51:

h(51)=1 => Vị trí này đã bị xóa

➔Băm lại: H(51, 1)=2 => Tìm thấy 51

	Kết quả
0	30
1	~~41~~
2	51
3	62
4	
5	45
6	
7	
8	
9	

	Kết quả
0	30
1	DELETED
2	51
3	62
4	
5	45
6	
7	
8	
9	

Xóa trong Linear Probing HashTable

- Thêm 71:

h(71)=1 => Vị trí này đã bị xóa

=> Thêm 71 vào vị trí này.

```
Insert(HashTable H, int x)
{
    while (H[h(x)] == occupied)
    {
        f(i);
    }
    H[h(x)] = x;
}
```

	Kết quả
0	30
1	DELETED
2	51
3	62
4	
5	45
6	
7	
8	
9	

	Kết quả
0	30
1	71
2	51
3	62
4	
5	45
6	
7	
8	
9	

Linear Probing Method: Nhận xét

- Bảng băm này chỉ tối ưu khi băm đều, nghĩa là trên bảng băm các khối đặc chứa vài phần tử và các khối phần tử chưa sử dụng xen kẽ nhau, tốc độ truy xuất lúc này có bậc O(1).
- Trường hợp xấu nhất là băm không đều hoặc bảng băm đầy, lúc này hình thành một khối đặc có n phần tử, nên tốc độ truy xuất lúc này có bậc O(n).

Linear Probing Method: Nhận xét (tt)

	Worst Case			Average Case		
Implementation	Search	Insert	Delete	Search	Insert	Delete
Sorted array	logN	N	N	logN	N/2	N/2
Unsorted array	N	N	N	N/2	N	N/2
Direct Chaining	N	N	N	1*	1*	1*
Linear Probing	N	N	N	1*	1*	1*

*assumes hash function is random

translate: gia dinh rang ham bam la ngau nhien

Quadratic Probing Method - PP dò bậc hai

- Hàm băm lại của phương pháp dò bậc hai là truy xuất các địa chỉ cách bậc 2. Hàm băm lại hàm i được biểu diễn bằng công thức sau:

$$H(key, i) = (h(key) + i^2) \% M$$

với h(key) là hàm băm chính của bảng băm, $f(i) = i^2$

- Nếu đã dò đến cuối bảng thì trở về dò lại từ đầu bảng.
- Bảng băm với phương pháp do bậc hai nên chọn số địa chỉ M là số nguyên tố.

Quadratic Probing Method

- Bảng băm dùng phương pháp dò tuyến tính bị hạn chế do rải các nút không đều, bảng băm với phương pháp dò bậc hai rải các nút đều hơn.
- Bảng băm trong trường hợp này được cài đặt bằng danh sách kế có M nút, mỗi nút của bảng băm là một mẩu tin có một trường key để chứa khóa các nút.
- Khi khởi động bảng băm thì tất cả trường key bị gán nullkey.
- Khi thêm nút có khóa key vào bảng băm, hàm băm h(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến M-1.

Quadratic Probing Method

- Nếu chưa bị xung đột thì thêm nút mới vào địa chỉ i.
- Nếu bị xung đột thì hàm băm lại lần 1 H(key, 1) sẽ xét địa chỉ cách 1^2, nếu lại bị xung đột thì hàm băm lại lần 2 H(key, 2) sẽ xét địa chỉ cách i 2^2,..., quá trình cứ thế cho đến khi nào tìm được trống và thêm nút vào địa chỉ này.
- Khi tìm môt nút có khóa key trong bảng băm thì xét nút tại địa chỉ i=h(key), nếu chưa tìm thấy thì xét nút cách i 1^2, 2^2, ..., quá trình cứ thế cho đến khi tìm được khóa (trường hợp tìm thấy) hoặc rơi vào địa chỉ trống (trường hợp không tìm thấy).

Quadratic Probing Method (dò bậc 2): Ví dụ (tt)

- Thêm: {41, 45, 51, 30, 62, 80, 53, 89, 77, 19}
- H(key, i) = (h(key) + i^2) % 10 , h(key) = key mod 10

Bài giải:

- h(51)=51 mod 10=1 => đụng độ

⇒Băm lại: H(51, 1) = (1 + 1^2) % 10=2

- h(30)=30 mod 10=0
- h(62)=2 => đụng độ

⇒Băm lại: H(62, 1) = (2 + 1^2) % 10=3

- h(80)=80 mod 10=0 => đụng độ

⇒Băm lại: H(80, 1) = (0 + 1^2) % 10=1 => đụng độ

⇒Băm lại: H(80, 2) = (0 + 2^2) % 10=4 => đụng độh(53)=3 => đụng độ

⇒ Băm lại lần 1: H(53, 1) = (3 + 1^2) % 10=4 => đụng độ

⇒ Băm lại lần 2: H(53, 2) = (3 + 2^2) % 10=7

- h(19)=19 mod 10=9 => đụng độ

=> Băm lại: H(19, 1) = (9 + 1^2) % 10=0 => đụng độ

=> Băm lại: H(19, 2) = (9 + 2^2) % 10=3 => đụng độ

⇒Băm lại: H(19, 3) = (9 + 3^2) % 10=8 => đụng độ

⇒ Băm lại: H(19, 4) = (9 + 4^2) % 10=5 => đụng độ

⇒Băm lại: H(19, 5) = (9 + 5^2) % 10=4 => đụng độ

⇒Băm lại: H(19, 6) = (9 + 6^2) % 10=5 => đụng độ

⇒Băm lại: H(19, 7) = (9 + 7^2) % 10=8 => đụng độ

⇒…

	Kết quả
0	30
1	41
2	51
3	62
4	80
5	45
6	
7	53
8	77
9	89

Quadratic Probing Method (băm dò bậc 2): Ví dụ

- Giả sử, khảo sát bảng băm có cấu trúc như sau:
  - Tập khóa K: tập số tự nhiên
  - Tập địa chỉ M: gồm 10 địa chỉ (M={0, 1, …, 9})
  - Hàm băm h(key) = key % 10.
- Hình thể hiện thêm các giá trị 10, 15, 16, 20, 30, 25, 26, 36 vào bảng băm.

Quadratic Probing Method: Ví dụ (tt)

Thêm vào các khóa 10, 15, 16, 20, 30, 25, 26, 36:

$$M=10,\ h(key)= key\%10,\ H(key, i) = (h(key) + i^2)\ \%\ 10$$

(Chú ý: Khi làm bài thi cần trình bày rõ vì sao thêm khóa k vào vị trí nào i trong bảng băm)

	Empty Table	**Thêm 10**	**Thêm 15**	**Thêm 16**	**Thêm 20**	**Thêm 30**	**Thêm 25**	**Thêm 26**	**Thêm 36**
0		10	10	10	10	10	10	10	10
1					20	20	20	20	20
2									36
3									
4						30	30	30	30
5			15	15	15	15	15	15	15
6				16	16	16	16	16	16
7								26	26
8									
9							25	25	25

Quadratic Probing Method: Nhận xét

- Nên chọn số địa chỉ M là số nguyên tố. Khi khởi động bảng băm thì tất cả M trường key được gán nullkey, biến toàn cục N được gán 0.
- Bảng băm đầy khi N = M-1, và nên dành ít nhất một phần tử trống trên bảng băm.
- Bảng băm này tối ưu hơn bảng băm dùng phương pháp dò tuyến tính do rải rác phần tử đều hơn, nếu bảng băm chưa đầy thì tốc độ truy xuất có bậc O(1). Trường hợp xấu nhất là bảng băm đầy vì lúc đó tốc độ truy xuất chậm do phải thực hiện nhiều lần so sánh.

Double Hash Method - PP Băm kép

- Hàm băm lại lần i được biểu diễn bằng công thức sau:

$$H(key, i) = (h1(key) + i * h2(key)) \% M$$

với h1(key) là hàm băm chính của bảng băm, f(i)= i* h2(key)

- Nếu đã dò đến cuối bảng thì trở về dò lại từ đầu bảng.
- Bảng băm với phương pháp băm kép nên chọn số địa chỉ M là số nguyên tố.

Double Probing Method - PP Băm kép

- Bảng băm này dùng hai hàm băm khác nhau với mục đích để rải rác đều các phần tử trên bảng băm.
- Chúng ta có thể dùng hai hàm băm bất kỳ, ví dụ chọn hai hàm băm như sau:

$$h1(key) = key \% M$$

$$h2(key) = R - key \% R$$ (thường chọn R là số nguyên tố nhỏ hơn M)

- Bảng băm trong trường hợp này được cài đặt bằng danh sách kề có M phần tử, mỗi phần tử của bảng băm là một mẫu tin có một trường key để lưu khoá các phần tử.

Double Probing Method

- **Khi khởi động:** bảng băm, tất cả trường key được gán nullkey.
- **Khi thêm phần tử:** có khoá key vào bảng băm, thì **i=h1(key)** và **j=h2(key)** sẽ xác định địa chỉ i và j trong khoảng từ 0 đến M-1:
  - Nếu chưa bị xung đột thì thêm phần tử mới tại địa chỉ i này.
  - Nếu bị xung đột thì hàm băm lại lần 1 f1 sẽ xét địa chỉ mới i+j, nếu lại bị xung đột thì hàm băm lại lần 2 là f2 sẽ xét địa chỉ i+2j, …, quá trình cứ thế cho đến khi nào tìm được địa chỉ trống và thêm phần tử vào địa chi này.

Double Probing Method: Ví dụ

Thêm vào các khóa 89, 18, 49, 58, 69:

M=10, h1(key)= key%10, h2(key) = 7 - key%7

H(key, i) = (h1(key) + i* h2(key)) % M

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

(Chú ý: Khi làm bài thi cần trình bày rõ vì sao thêm khóa k vào vị trí nào i trong bảng băm)

Double Probing Method

- **Khi tìm kiếm** một phần tử có khoá key trong bảng băm, hàm băm **i=h1(key)** và **j=h2(key)** sẽ xác định địa chỉ i và j trong khoảng từ 0 đến M-1. Xét phần tử tại địa chỉ i, nếu chưa tìm thấy thì xét tiếp phần tử i+j, i+2j, …, quá trình cứ thế cho đến khi nào tìm được khoá (trường hợp tìm thấy) hoặc bị rơi vào địa chỉ trống (trường hợp không tìm thấy).

Double Probing Method: Nhận xét

- Nên chọn số địa chỉ M là số nguyên tố.
- Bảng băm đầy khi N = M-1, chúng ta nên dành ít nhất một phần tử trống trên bảng băm.
- Bảng băm được cài đặt theo cấu trúc này linh hoạt hơn bảng băm dùng phương pháp dò tuyến tính và bảng băm dùng phương pháp sò bậc hai, do dùng hai hàm băm khác nhau nên việc rải phần tử mang tính ngẫu nhiên hơn, nếu bảng băm chưa đầy tốc độ truy xuất có bậc O(1). Trường hợp xấu nhất là bảng băm gần đầy, tốc độ truy xuất chậm do thực hiện nhiều lần so sánh.

Thao tác Insert

- ***Đầu vào***:
  - Bảng băm T có kích thước m,
  - Hàm băm lại hi(k).
  - Phần tử có khóa k cần thêm
- ***Đầu ra***:
  - Bảng băm T.
  - Vị trí của phần tử được chèn hoặc -1 nếu gặp lỗi.
- Ký hiệu DELETED và NIL được dùng để đánh dấu vị trí đã được xóa và vị trí trống

Thao tác Insert

```
HASH-INSERT(T, k)
 1. i = 0
 2. do
 3.     j = h(k, i)
 4.     if T[j] == NIL or T[j] == DELETED
 5.         T[j] = k
 6.         return j
 7.     else i = i + 1
 8. while i < m
 9. return -1
```

Thao tác xóa phần tử

- ***Đầu vào***:
  - Bảng băm T có kích thước m,
  - Hàm băm lại hi(k).
  - Khóa k cần xóa.
- ***Đầu ra***:
  - Bảng băm T.
  - Vị trí của phần tử được xóa hoặc -1 nếu gặp lỗi.

Ký hiệu DELETED và NIL được dùng để đánh dấu vị trí đã được xóa và vị trí trống.

Thao tác xóa phần tử

```
HASH-DELETE(T, k)
 1.  i = 0
 2.  do
 3.        j = h(k, i)
 4.        if T[j] == k
 5.            temp = T[j]
 6.            T[j] = DELETED
 7.            return temp
 8.        else i = i + 1
 9.  while T[j] != NIL and i < m
10.  return -1
```

Thao tác tìm phần tử

- ***Đầu vào***:
  - Bảng băm T có kích thước m,
  - Hàm băm lại hi(k).
  - Khóa k cần tìm.
- ***Đầu ra***:
  - Vị trí của phần tử được tìm hoặc -1 nếu gặp lỗi.

Ký hiệu DELETED và NIL được dùng để đánh dấu vị trí đã được xóa và vị trí trống.

Thao tác tìm phần tử

```
HASH-SEARCH(T, k)
1. i = 0
2. do
3.  j = h(k, i)
4.  if T[j] == k
5.      return j
6.  else i = i + 1
7. while T[j] != NIL and i < m
8. return -1
```

Lưu ý về Load Factor khi băm lại

- Băm lại (**rehashing**) cần được thực hiện khi:
  - Không thể thực hiện một thao tác thêm phần tử mới bất kỳ.
  - Bảng băm có hệ số tải đạt một giá trị xác định ($\lambda \geq 0.5$) làm độ phức tạp thời gian của thao tác tìm kiếm tăng.

- Các bước thực hiện khi băm lại:
  - Tăng kích thước bảng băm lên m’ với m’ là một số nguyên tố và $m' \approx 2 * m$.
  - Cập nhật hàm băm.
  - Băm lại các phần tử có trong bảng băm cũ và thêm vào bảng băm mới.

Các lưu ý trong quá trình Băm lại (Rehashing)

- Ví dụ: Thêm phần tử có khóa 23 vào bảng băm sau với hệ số tải của bảng băm là 0.5:

0	1	2	3	4	5	6
6	15		24			13

- Hệ số tải λ = 4/7 =0.57 > 0.5 ➔ Băm lại
- Tăng kích thước bảng băm lên m’ = 17

- Hàm băm mới: h’(k) = k mod 17
- Băm lại các giá trị 6, 15, 24, 13 vào bảng băm mới, sử dụng hàm băm mới
- Băm phần tử 23 vào bảng băm mới

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
						6	23	24					13		15	

Thư viện unordered_map

- Chèn #include <unordered_map>
- unordered_map là một tập hợp các phầntử được truy xuất như bảng băm theo thuộc tính khóa.
- Mỗi phần tử gồm một cặp key, value được đặt trong kiểu pair<key, value>
- Mỗi biến kiểu pair có trường pair.first và pair.second lần lượt là trường key và value của một phần tử.
- Các thao tác cơ bản của unordered_map gồm: insert – thêm phần tử, find – tìm phần tử có khóa k, erase – xóa phần tử.
- ***Sinh viên tìm hiểu thêm về unordered_map***

Thư viện unordered_map

- Ví dụ: Viết chương trình nhập vào thông tin sinh viên gồm mã số sinh viên và tên (không chứa khoảng trắng); cho phép tìm thông tin theo mã số sinh viên, in ra màn hình và xóa thông tin sinh viên vừa tìm được

Thư viện unordered_map

```cpp
#include <iostream>
#include <string>
#include <unordered_map>
using namespace std;
unordered_map<int, string> nhap();
void tim(unordered_map<int, string> &, int);
int main() {
    unordered_map<int, string> ds = nhap();
    int ms;
    cin >> ms;
    tim(ds, ms);
    return 0;
}
```

Thư viện unordered_map

```
unordered_map<int, string> nhap(){
    unordered_map<int, string> ds;
    pair<int, string> hs;
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> hs.first >> hs.second;
        ds.insert(hs);
    }
    return ds;
}
```

Thư viện unordered_map

```
void tim(unordered_map<int, string> &ds, int ms){
    unordered_map<int, string>::iterator
    iter = ds.find(ms);
    if (iter != ds.end()) {
        cout << "MSSV: " << (*iter).first
        << "Ten: " << (*iter).second
        << '\n';
        ds.erase(iter);
    }
    else
        cout << "Khong tim thay\n";
}
```

Thư viện unordered_set

- #include <unordered_set>
- unordered_set là một tập hợp các phần tử được truy xuất như bảng băm.
- Mỗi phần tử là một biến có kiểu dữ liệu bất kỳ
- Các thao tác cơ bản của unordered_set gồm: insert – thêm phần tử, find – tìm phần tử, erase – xóa phần tử.
- ***Sinh viên tìm hiểu thêm về unordered_set***

Thư viện unordered_set

- Ví dụ: Viết chương trình nhập vào một dãy số nguyên; cho phép tìm một số nguyên k trong dãy đã nhập, in ra màn hình “YES” nếu tìm thấy và “NO” nếu không tìm thấy số cần tìm, sau đó xóa số vừa tìm được.

Ví dụ: Thư viện unordered_set

```
#include <iostream>
#include <string>
#include <unordered_set>
using namespace std;
unordered_map<int> nhap();
void tim(unordered_set<int> &, int);
int main() {
    unordered_set<int> ds = nhap();
    int k;
    cin >> k;
    tim(ds, k);
    return 0;
}
```

Ví dụ: Thư viện unordered_set

```
unordered_set<int> nhap(){
    unordered_set<int> ds;
    int n, tmp;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> tmp;
        ds.insert(tmp);
    }
    return ds;
}
```

Ví dụ: Thư viện unordered_set

```
void tim(unordered_set<int> &ds, int k){
    unordered_set<int>::iterator iter=ds.find(k);
    if (iter != ds.end()) {
        cout << "YES\n";
        ds.erase(iter);
    }
    else
        cout << "NO\n";
}
```

Bài tập

- Bài 1: Hãy mô tả các bước xảy ra khi chèn các khoá 5, 28, 19, 15, 20, 33, 12, 17, 10 vào một bảng băm được giải quyết đụng độ bằng phương pháp nối kết. Cho bảng băm có 9 ô và hàm băm là h(k) = k mod 9
- Bài 2: Xét một bảng băm có kích thước là m=1000 và hàm băm h(k)=**h(k) = floor(m(kA mod 1))** với A=. A = ½(Ö5 -1) ≈ 0.6180339887. Hãy tính vị trí các khóa 61, 62, 63, 65
- Bài 3: Thêm các khoá 10, 22, 31, 4, 15, 28, 17, 88, 59 vào một bảng băm có kích thước m = 11 sử dụng địa chỉ mở với hàm băm h(k) = k mod 11. Hãy minh họa kết quả khi thêm các khóa này vào bảng băm sử dụng phương pháp dò tuyến tính, dò bậc hai, và băm kép với hàm băm phụ là h'(k) = 1 + (k mod (m-1))

Câu hỏi và Bài tập

Bài 4. Cho cỡ bảng băm SIZE = 11. Từ bảng băm rỗng, sử dụng hàm băm chia lấy dư, hãy đưa lần lượt các dữ liệu với khoá:

32 , 15 , 25 , 44 , 36 , 21

vào bảng băm và đưa ra bảng băm kết quả trong các trường hợp sau:

a. Bảng băm được chỉ mở với thăm dò tuyến tính.

b. Bảng băm được chỉ mở với thăm dò bình phương.

c. Bảng băm dây chuyền.

Bài 5. Từ các bảng băm kết quả trong bài tập 4, hãy loại bỏ dữ liệu với khoá là 44 rồi sau đó xen vào dữ liệu với khoá là 65.

VI. BÀI TẬP

- Bài 6: Thêm các khoá 10, 22, 31, 4, 15, 28, 17, 88, 59 vào một bảng băm có kích thước m = 11 sử dụng địa chỉ mở với hàm băm h(k) = k mod 11. Hãy minh họa kết quả khi thêm các khóa này vào bảng băm sử dụng phương pháp dò tuyến tính, dò bậc hai, và băm kép với hàm băm phụ là h'(k) = 1 + (k mod (m-1))

Chúc các em học tốt!