ĐẠI HỌC QUỐC GIA TPHCM
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

CHƯƠNG V

BẢNG BĂM

Nguyễn Trọng Chỉnh
chinhnt@uit.edu.vn

BẢNG BĂM

- **CÁC KHÁI NIỆM**
- PHƯƠNG PHÁP NỐI KẾT
- PHƯƠNG PHÁP ĐỊA CHỈ MỞ

CÁC KHÁI NIỆM

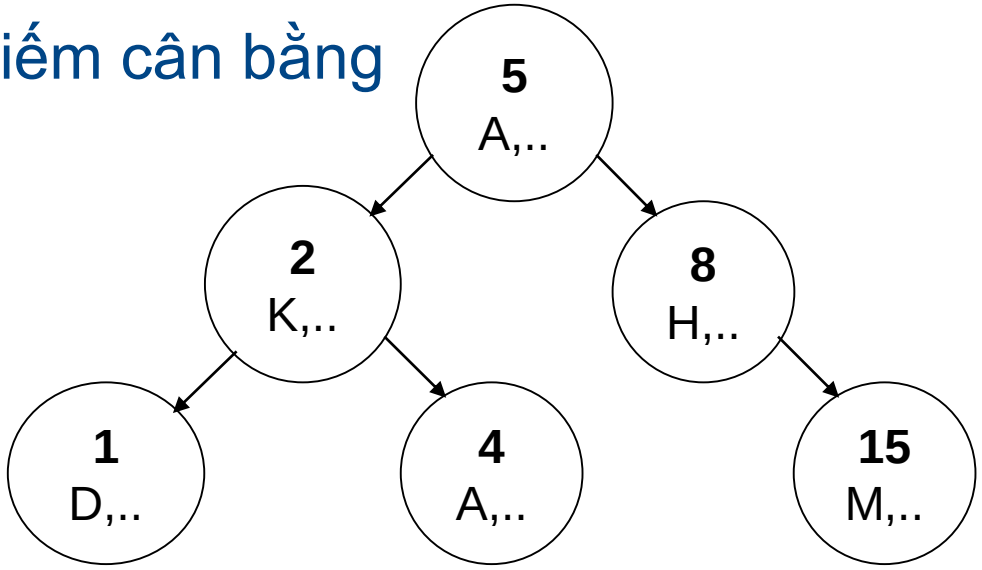
❖BẢNG BĂM (Hashtable)

Vấn đề: cho tập dữ liệu gồm n nhân viên, mỗi nhân viên gồm thông tin mã số nhân viên, họ tên, mức lương, việc tìm kiếm được thực hiện trên trường mã số (trường khóa). Xét các cách lưu trữ sau:

1) Dùng cây nhị phân tìm kiếm cân bằng

Thời gian truy xuất:

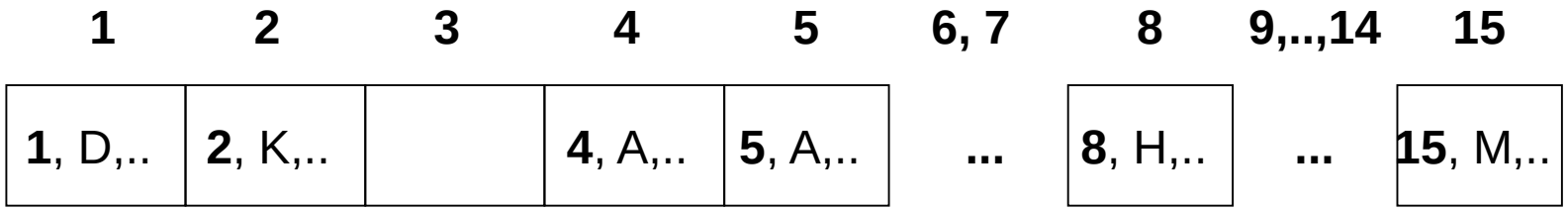
O(log(n))



CÁC KHÁI NIỆM

❖BẢNG BĂM (Hashtable)

2) Dùng mảng sao cho chỉ số của mảng trùng với mã số nhân viên



thời gian truy xuất O(1)

Nhược điểm:

- Miền giá trị của mã số lớn thì không thể tạo mảng
- Lãng phí bộ nhớ
- Chỉ áp dụng cho dữ liệu số nguyên

CÁC KHÁI NIỆM

❖BẢNG BĂM (Hashtable)

Khái niệm bảng băm: là một cấu trúc dữ liệu tương tự mảng có kích thước m, cho phép ánh xạ một giá trị khóa thành một địa chỉ trong đoạn [0, m-1] nhằm xác định nhanh chóng phần tử có khóa cần xử lý.

Ví dụ 1: để lưu trữ thông tin nhân viên trên, dùng bảng băm với hàm băm là f(x) = x % 9

1	2	3	4	5	6	7	8
1, D,..	**2**, K,..		**4**, A,..	**5**, A,..	**15**, M,..		**8**, H,..

CÁC KHÁI NIỆM

❖HÀM BĂM (Hash function)

Ánh xạ biến giá trị khóa thành địa chỉ trong bảng băm

Tính chất của hàm băm:

- Phải trả về giá trị trong đoạn [0, m-1]
- Tính toán đơn giản với độ phức tạp O(1)
- Không lãng phí không gian. Với mỗi địa chỉ trong bảng băm, phải có ít nhất 1 khóa có giá trị hàm băm bằng nó.
- Tối thiểu hóa đụng độ. Các khóa khác nhau sẽ có giá trị hàm băm ít trùng nhau.

CÁC KHÁI NIỆM

❖ĐỤNG ĐỘ (Collision)

Giả sử có hàm băm f(k), đụng độ là hiện tượng có hai khóa k_1 và k_2 không trùng nhau nhưng $f(k_1) = f(k_2)$

Ví dụ 2: cho bảng băm kích thước 6, các giá trị khóa là 2, 4, 6, 8, 10. Xét hàm băm f(k) = k % 6:

- Có f(2)=2, f(4)=4, f(6)=0, f(8)=2, f(10)=4, f(12)=0: hàm f(k) không xác định được các vị trí 1, 3, 5 $\rightarrow$ chưa đảm bảo tính chất của hàm băm
- Có f(2)=f(8), f(4)=f(10), f(6)=f(12): hàm f(k) chưa tối thiểu hóa được các đụng độ.

CÁC KHÁI NIỆM

❖CÁC DẠNG HÀM BĂM

- **Hàm băm dạng bảng**: các khóa được liệt kê tương ứng với địa chỉ.

Ví dụ 3: có f(k) được cho như bảng sau:

khóa	for	long	to	do	fix	int	byte	in	if	while
địa chỉ	0	1	2	3	4	5	6	7	8	9

có: f("if") = 8,f("fix") = 4.

CÁC KHÁI NIỆM

❖CÁC DẠNG HÀM BĂM

- **Hàm băm dùng phương pháp chia**: là hàm băm có dạng:

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

trong đó m là kích thước của bảng băm, k là khóa có giá trị nguyên.

Để tránh hiện tượng như ví dụ 2: kích thước bảng băm m sẽ được chọn là số nguyên tố nhỏ nhất và lớn hơn hoặc bằng kích thước bảng băm cần thiết.

CÁC KHÁI NIỆM

❖CÁC DẠNG HÀM BĂM

- **Hàm băm dùng phương pháp chia**:

Ví dụ 4: Với yêu cầu bảng băm chứa 6 phần tử có khóa là 2, 4, 6, 8, 10, 12, kích thước cần cho bảng băm là 7, hàm băm dùng phương pháp chia là:

$$f(k) = k \% 7$$

Khi đó:

f(2) = 2, f(4) = 4, f(6) = 6, f(8) = 1, f(10) = 3, f(12) = 5.

CÁC KHÁI NIỆM

❖CÁC DẠNG HÀM BĂM

- **Hàm băm dùng phương pháp chia:**

  Trường hợp khóa có dạng chuỗi: dùng công thức Horner với m là kích thước bảng băm, n là chiều dài khóa k:

  $h_{n-1} = k[n-1] \% m$

  $h_{n-2} = (k[n-2] + h_{n-1}*32) \% m$

  $h_{n-3} = (k[n-3] + h_{n-2}*32) \% m$

  ....

  $h_0 = (k[0] + h_1*32) \% m$

  $f(k) = h_0$

CÁC KHÁI NIỆM

❖CÁC DẠNG HÀM BĂM

- **Hàm băm dùng phương pháp chia**:

ví dụ 5: cho bảng băm kích thước 7, dùng hàm băm theo công thức Horner tính địa chỉ cho chuỗi "for"

h_2 = 'r' % 7 = 114 % 7 = 2

h_1 = ('o' + 2*32) % 7 = (111 + 64) % 7 = 0

h_0 = ('f' + 0*32) % 7 = 102 % 7 = 4

$\rightarrow$ f("for") = 4.

CÁC KHÁI NIỆM

❖CÁC DẠNG HÀM BĂM

- **Hàm băm dùng phương pháp nhân**: là hàm băm có dạng:

$$f(k) = \lfloor m * \{k * A\} \rfloor$$

- m là kích thước của bảng băm, m thường chọn là 2^p
- k là khóa.
- A là hằng số tùy chọn trong khoảng (0, 1). Knuth đề nghị $A = (\sqrt{5} + 1) / 2$.
- {} là phép toán lấy phần thập phân.

CÁC KHÁI NIỆM

❖CÁC DẠNG HÀM BĂM

- **Hàm băm dùng phương pháp nhân**:

Ví dụ 6: cho bảng băm kích thước 8, hàm băm dùng phương pháp nhân với hằng số A do Knuth đề nghị, tính địa chỉ của khóa k = 110

$f(k) = \lfloor 8 * \{110 * ((\sqrt{5} + 1) / 2)\} \rfloor$

$= \lfloor 8 * \{177.984\} \rfloor = \lfloor 8 * 0.984 \rfloor = \lfloor 7.86991 \rfloor$

$= 7$

Lưu ý: trong C/C++, để lấy phần thập phân của một số thực x, dùng hàm: fmod(x, 1) được khai báo trong cmath

PHƯƠNG PHÁP NỐI KẾT

❖ PHƯƠNG PHÁP NỐI KẾT (CHAINING)

Phương pháp này giải quyết đụng độ bằng cách tạo một danh sách các phần tử có địa chỉ trùng nhau. Bảng băm sẽ dùng danh sách liên kết đơn tại mỗi địa chỉ của nó để nối kết các phần tử trong trường hợp đụng độ.

PHƯƠNG PHÁP NỐI KẾT

❖PHƯƠNG PHÁP NỐI KẾT (CHAINING)

Phương pháp nối kết có 2 dạng:

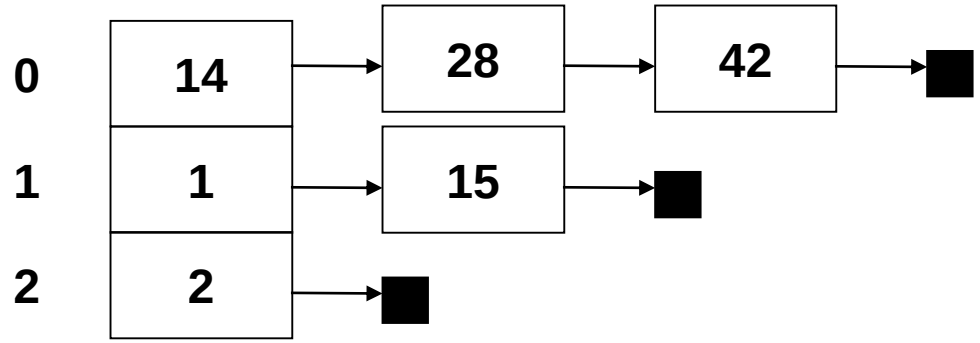
+ Nối kết trực tiếp (direct chaining): khi gặp đụng độ sẽ tạo một phần tử mới và nối vào danh sách tại vị trí có đụng độ

+ Nối kết hợp nhất (coalesced chaining): khi gặp đụng độ sẽ dùng phần tử trống đầu tiên tính từ cuối mảng để nối vào danh sách tại vị trí có đụng độ.

PHƯƠNG PHÁP NỐI KẾT

❖PHƯƠNG PHÁP NỐI KẾT (CHAINING)

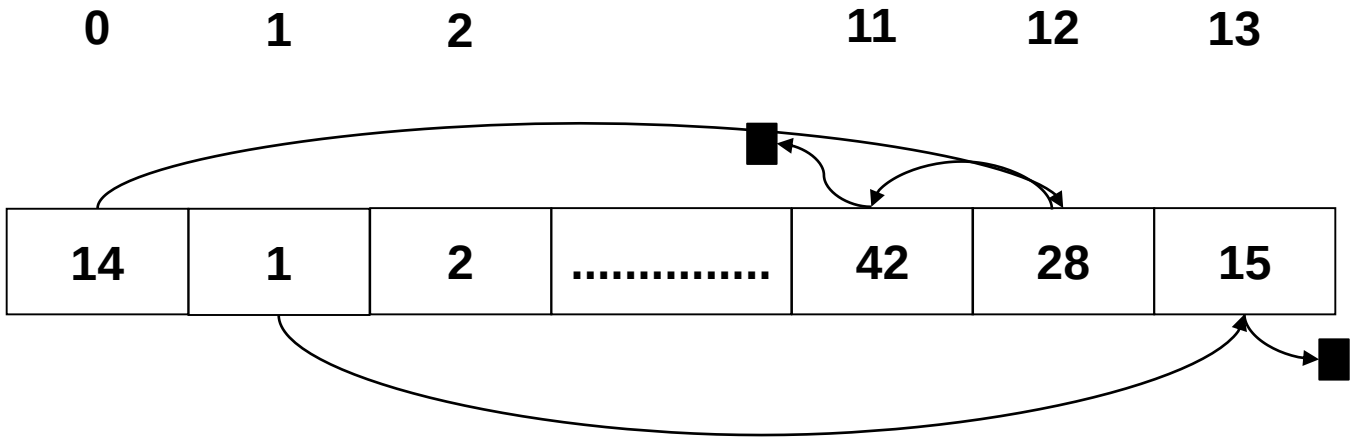
Ví dụ 7: Cho bảng băm dùng phương pháp nối kết trực tiếp với số phần tử là 14, sử dụng hàm băm theo phương pháp chia. Các khóa cần lưu trữ là 1, 2, 14, 15, 28, 42.



PHƯƠNG PHÁP NỐI KẾT

❖PHƯƠNG PHÁP NỐI KẾT (CHAINING)

Ví dụ 8: Cho bảng băm dùng phương pháp nối kết hợp nhất với số phần tử là 14, sử dụng hàm băm theo phương pháp chia. Các khóa cần lưu trữ là 1, 2, 14, 15, 28, 42.



PHƯƠNG PHÁP NỐI KẾT

❖CẤU TRÚC DỮ LIỆU

```
struct Node {
    int key; // giả sử chỉ có thông tin khóa k có kiểu int
    Node * pNext;
};
struct LIST {      Node * pHead, * pTail;  };
struct OHashtable {
    int m;
    LIST *buckets;
};
```

PHƯƠNG PHÁP NỐI KẾT

❖CÁC THAO TÁC

- Tạo bảng băm kích thước m (nối kết trực tiếp)

```
void CreateList(LIST &l);
void CreateOHashtable(OHashtable &ht, int m) {
    ht.buckets = new LIST[m];
    if (ht.buckets == NULL) ht.m = 0;
    else {
    for (int i = 0; i < m; i++) CreateList(ht.buckets[i]);
    ht.m = m;
    }
}
```

PHƯƠNG PHÁP NỐI KẾT

❖CÁC THAO TÁC

- Đưa một phần tử có khóa x (nối kết trực tiếp)

```
Node * CreateNode(int key); // tạo node có giá trị k
void AddLast(LIST &l, Node *p); // thêm cuối danh sách
int OHashCode(int key, int m); // cài đặt dạng chia hoặc nhân
int OPutKey(OHashtable ht, int key) {
    if (ht.m == 0) return 0;
    Node * p = CreateNode(key);
    if (p == NULL) return 0;
    int i = OHashCode(key, ht.m); AddLast(ht.buckets[i], p);
    return 1; }
```

PHƯƠNG PHÁP NỐI KẾT

❖CÁC THAO TÁC

- Lấy phần tử có khóa x (nối kết trực tiếp)

```
Node * Search(LIST l, int x); // hàm tìm node có khóa x
Node * OGetKey(OHashtable ht, int key) {
    if (ht.m == 0) return NULL;
    int i = OHashCode(key, ht.m);
    return Search(ht.buckets[i], key);
}
```

PHƯƠNG PHÁP NỐI KẾT

❖CÁC THAO TÁC

- Xóa phần tử có khóa x (nối kết trực tiếp)

```
int Remove(LIST &l, int x); // hàm xóa node có khóa x
int ORemoveKey(OHashtable ht, int key) {
    if (ht.m == 0) return 0;
    int i = OHashCode(key, ht.m);
    return Remove(ht.buckets[i], key);
}
```

PHƯƠNG PHÁP NỐI KẾT

❖CÁC THAO TÁC

- Hủy toàn bộ bảng băm (nối kết trực tiếp)

```
void RemoveList(LIST &l); // hàm hủy toàn bộ danh sách
void RemoveOHashtable(OHashtable &ht) {
    for (int i = 0; i < ht.m; i++) {
    RemoveList(ht.buckets[i]);
    ht.buckets[i] = NULL;
    }
    delete[] ht.buckets;
    ht.m = 0;
}
```

PHƯƠNG PHÁP NỐI KẾT

❖ĐẶC ĐIỂM

- Số phần tử không cố định
- Một số khóa có thể có cùng địa chỉ
- Có thể thực hiện thao tác thêm, xóa phần tử
- Thời gian truy xuất bị suy giảm.

PHƯƠNG PHÁP NỐI KẾT

❖BÀI TẬP

Cho mảng dữ liệu gồm các khóa có giá trị nguyên, xây dựng bảng băm theo phương pháp nối kết trực tiếp có kích thước m với hàm băm theo phương pháp chia. Viết chương trình tạo hàm băm có thể lưu 8 phần tử và nhập một dãy khóa nguyên vào bảng băm đó. In toàn bộ bảng băm và tìm và in ra phần tử có khóa x được nhập từ bàn phím.

BẢNG BĂM DẠNG MỞ

```
struct Node {
    int key;
    Node * pNext;
};
struct LIST {
    Node *pHead, *pTail;
};
struct OHashtable {
    int m;
    LIST *buckets;
};
```

BẢNG BĂM DẠNG MỞ

```
void CreateList(LIST &l) {
    l.pHead = NULL; l.pTail = NULL;
}
Node *CreateNode(int x) {
    Node * p = new Node;
    if (p!= NULL) {
        p->key = x;
        p->pNext = NULL;
    }
    return p;
}
```

BẢNG BĂM DẠNG MỞ

```
void AddLast(LIST &l, Node *p) {
    if (l.pHead == NULL) {
        l.pHead = p; l.pTail = p;
    }
    else {
        l.pTail->pNext = p; l.pTail = p;
    }
}
Node * Search(LIST &l, int x) {
    Node *p = l.pHead;
    while ((p!= NULL) && (p->key != x))
        p = p->pNext;
    return p;
}
```

BẢNG BĂM DẠNG MỞ

```
void CreateOHashtable(OHashtable &ht, int m) {
    ht.buckets = new LIST[m];
    if (ht.buckets == NULL)
        m = 0;
    else {
        for (int i = 0; i < m; i++)
            CreateList(ht.buckets[i]);
        ht.m = m;
    }
}
int OHashCode(int key, int m) {
    return key % m;
}
```

BẢNG BĂM DẠNG MỞ

```
int OPutKey(OHashtable ht, int key) {
    if (ht.m == 0) return 0;
    int i = OHashCode(key, ht.m);
    Node *p = CreateNode(key);
    if (p == NULL) return 0;
    AddLast(ht.buckets[i], p);
    return 1;
}
Node *OGetKey(OHashtable ht, int key) {
    if (ht.m == NULL) return NULL;
    int i = OHashCode(key, ht.m);
    return Search(ht.buckets[i], key);
}
```

BẢNG BĂM DẠNG MỞ

```
int GetTableSize(int m) {
    int n = 1, k, i;
    double x;
    while (n < m) n = n * 2;
    n = n + 1;
    k = m;
    do {
        x = sqrt((double) k);
        for (i = 2; i < x; i++)
            if ((k % i) == 0) break;
        if (i < x) k++;
    }while ((x > i) && (k < n));
    return k;
}
```

BẢNG BĂM DẠNG MỞ

```
void print(OHashtable ht) {
    int i;
    Node *p;
    if (ht.m == 0) return;
    for (i = 0; i < ht.m; i++) {
        p = ht.buckets[i].pHead;
        while (p != NULL) {
            cout << p->key << ' ';
            p = p->pNext;
        }
        cout << endl;
    }
}
```

BẢNG BĂM DẠNG MỞ

```
void main() {
    OHashtable ht;
    int x, n;
    cout << "Nhap kich thuoc bang bam ";
    cin >> n;
    n = GetTableSize(n);
    CreateOHashtable(ht, n);
    cout << "So luong khoa can nhap ";
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> x;
        if (!OPutKey(ht, x)) return;
    }
```

BẢNG BĂM DẠNG MỞ

```
    print(ht);
    cout << "nhap khoa can tim ";
    cin >> x;
    Node * p = OGetKey(ht, x);
    if (p == NULL)
        cout << "khong tim thay" << endl;
    else
        cout << "tim thay " << p->key << endl;
}
```

PHƯƠNG PHÁP ĐỊA CHỈ MỞ

❖ PHƯƠNG PHÁP ĐỊA CHỈ MỞ (Open Addressing)

Phương pháp này giải quyết đụng độ bằng thăm dò (probe) từng bước những địa chỉ khác còn trống trên bảng băm kích thước m nhờ một hàm băm thăm dò.

Hàm băm thăm dò f có dạng như sau:

$$f(k, i) = (h(k) + g(k, i)) \% m$$

- h(k) là một hàm băm.
- g(k, i) là một hàm số theo i, k
- i là lần thăm dò

PHƯƠNG PHÁP ĐỊA CHỈ MỞ

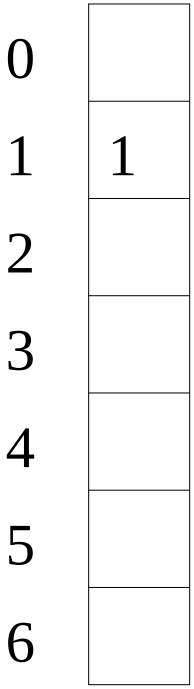
❖ PHƯƠNG PHÁP ĐỊA CHỈ MỞ (Open Addressing)

Ví dụ 9: Cho bảng băm dạng đóng kích thước 7, hàm băm thăm dò f(k, i) = ((k % m) + i) % m, Các khóa được đưa vào lần lượt là 1,2,8,15.

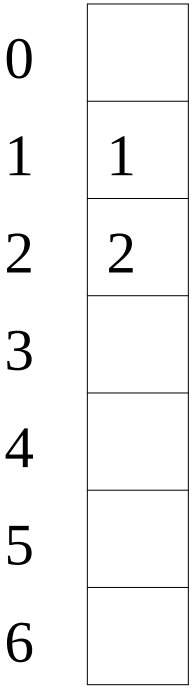
PHƯƠNG PHÁP ĐỊA CHỈ MỞ

❖PHƯƠNG PHÁP ĐỊA CHỈ MỞ (Open Addressing)

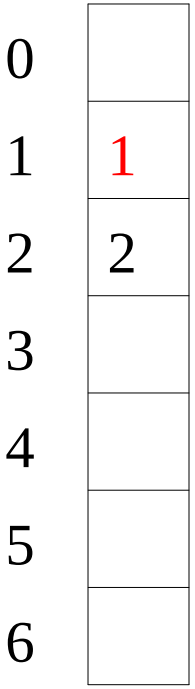
1, lần thăm dò 0
f(1, 0) = 1



2, lần thăm dò 0
f(2, 0) = 2



8, lần thăm dò 0
f(8, 0) = 1



PHƯƠNG PHÁP ĐỊA CHỈ MỞ

❖PHƯƠNG PHÁP ĐỊA CHỈ MỞ (Open Addressing)

8, lần thăm dò 1
f(8, 1) = 2

0	
1	1
2	2
3	
4	
5	
6	

8, lần thăm dò 2
f(8, 2) = 3

0	
1	1
2	2
3	8
4	
5	
6	

15, lần thăm dò 0
f(15, 0) = 1

0	
1	1
2	2
3	8
4	
5	
6	

PHƯƠNG PHÁP ĐỊA CHỈ MỞ

❖ PHƯƠNG PHÁP ĐỊA CHỈ MỞ (Open Addressing)

15, lần thăm dò 1
f(15, 1) = 2

0	
1	1
2	2
3	8
4	
5	
6	

15, lần thăm dò 2
f(15, 2) = 3

0	
1	1
2	2
3	8
4	
5	
6	

15, lần thăm dò 3
f(15, 2) = 4

0	
1	1
2	2
3	8
4	15
5	
6	

PHƯƠNG PHÁP ĐỊA CHỈ MỞ

❖CÁC PHƯƠNG PHÁP THĂM DÒ

- Phương pháp thăm dò tuyến tính: nếu hàm g(k, i) là một hàm tuyến tính theo i có dạng

$$g(k, i) = a*i + b.$$

Có thể chọn g(k, i) = i.

- Phương pháp thăm dò bậc 2 (toàn phương): nếu hàm g(k, i) là một hàm bậc 2 theo i có dạng

$$g(k, i) = a * i^2 + b * i + c.$$

Có thể chọn $g(k, i) = i^2$

PHƯƠNG PHÁP ĐỊA CHỈ MỞ

❖CÁC PHƯƠNG PHÁP THĂM DÒ

- Phương pháp băm kép: nếu g(k, i) là một hàm số có dạng

$$g(k, i) = i * h_1(k).$$

$h_1(k)$ được gọi là hàm băm phụ và cần được xây dựng theo kích thước bảng băm m như sau:

  - Trường hợp $m = 2^p$, $h_1(k)$ cần trả về giá trị lẻ.
  - Trường hợp m là số nguyên tố, $h_1(k)$ cần có miền giá trị là (0, m)

PHƯƠNG PHÁP ĐỊA CHỈ MỞ

❖CÁC PHƯƠNG PHÁP THĂM DÒ

Ví dụ 9: cho bảng băm đóng kích thước 7 dùng phương pháp băm kép, hàm băm phụ cần có dạng:

$$h_1(k) = 1 + (k \% 6)$$

Ví dụ 10: cho bảng băm đóng kích thước 8 dùng phương pháp băm kép, hàm băm phụ cần có dạng:

$$h_1(k) = (k\%8) + ((k+1)\%2)$$

PHƯƠNG PHÁP ĐỊA CHỈ MỞ

❖TỔ CHỨC DỮ LIỆU

```
#define DEL -1
#define EMPTY 0
// giả sử khóa có giá trị là số nguyên dương.
struct CHashtable {
    int m, n;
    int * buckets; // giả sử bảng băm chỉ lưu khóa
};
```

PHƯƠNG PHÁP ĐỊA CHỈ MỞ

❖CÁC THAO TÁC

- Tạo bảng băm đóng

```
void CreateCHashtable(CHashtable &ht, int m) {
    ht.buckets = new int[m];
    if (ht.buckets == NULL) m = 0;
    else {
    for (int i = 0; i < m; i++) ht.buckets[i] = EMPTY;
        ht.m = m;
    }
    ht.n = 0;
}
```

PHƯƠNG PHÁP ĐỊA CHỈ MỞ

❖CÁC THAO TÁC

- Đưa phần tử có khóa x vào bảng băm

```
int g(int key, int m, int i);    // hàm tuyến tính, bậc 2 hay hàm
                                 // băm phụ
int h(int key, int m);           // hàm băm dạng chia hay nhân
int CHashCode(int key, int m, int i) {
   return (h(key, m) + g(key, m, i)) % m;
}
```

PHƯƠNG PHÁP ĐỊA CHỈ MỞ

```
int CPutKey(CHashtable &ht, int key) {
    if (ht.m == 0 || ht.n == ht.m) return 0;
    int i = 0, k;
    do {
        k = CHashCode(key, ht.m, i); i++;
    } while (((ht.buckets[k] != EMPTY) ||
                (ht.buckets[k] != DEL)) && (i <= ht.m));
    if (i >= ht.m) return 0;
    ht.buckets[k] = key; ht.n++;
    return 1;
}
```

PHƯƠNG PHÁP ĐỊA CHỈ MỞ

❖CÁC THAO TÁC

- Lấy địa chỉ của phần tử có khóa x

```
int CGetKey(CHashtable ht, int key) {
    if (ht.m == 0) return -1;
    int i = 0, k;
    do {
        k = CHashCode(key, ht.m, i); i++;
    } while ((i <= ht.m) && (ht.buckets[k] != key) &&
                            (ht.buckets[k] != EMPTY));
    if (ht.buckets[k] == key) return k; else return -1;
}
```

PHƯƠNG PHÁP ĐỊA CHỈ MỞ

❖CÁC THAO TÁC

- Lấy phần tử tại bucket i

```
int CGetBucket(CHashtable ht, int i) {
    if (ht.m == 0) return EMPTY;
    return ht.buckets[i];
}
int isFull(CHashtable ht) {
    return ht.n == ht.m - 1;
}
```

PHƯƠNG PHÁP ĐỊA CHỈ MỞ

❖CÁC THAO TÁC

- Xóa phần tử có khóa x

```
void CRemoveKey(CHashtable &ht, int key) {
    if (ht.m == 0) return;
    int i = 0, k;
    do {
        k = CHashCode(key, ht.m, i); i++;
    } while ((i <= ht.m) && (ht.buckets[k] != key) &&
            (ht.buckets[k] != EMPTY));
    if ((i >= ht.m) || (ht.buckets[k] == EMPTY)) return;
    ht.buckets[k] = DEL; ht.n--; }
```

PHƯƠNG PHÁP ĐỊA CHỈ MỞ

❖CÁC THAO TÁC

- Hủy toàn bộ bảng băm

```
void RemoveCHashtable(CHashtable &ht) {
    if (ht.m == 0) return;
    delete[] ht.buckets;
    ht.buckets = NULL; m = 0;
}
```

PHƯƠNG PHÁP ĐỊA CHỈ MỞ

❖ĐẶC ĐIỂM

- Số phần tử cố định
- Mỗi khóa ứng với một địa chỉ
- Thời gian truy xuất thấp.
- Xóa một phần tử không thu hồi được vùng nhớ của nó.
- Luôn chừa 1 phần tử trống trong bảng băm, nghĩa là nếu bảng băm có kích thước m thì sẽ đầy khi số phần tử trong bảng băm là m-1.

LƯU Ý

Để tính quá trình tính toán địa chỉ trên bảng băm của một khóa k vào lần thăm dò thứ i được hiệu quả, cần cài đặt các thao tác thêm, tìm kiếm và xóa cụ thể cho từng phương pháp thăm dò. (xem Giáo trình)