Chương 3

Bảng băm (Hash table)

Nội dung

1. Bảng băm
2. Định nghĩa hàm băm
3. Phương pháp xây dựng hàm băm
4. Phương pháp giải quyết đụng độ

Bảng băm (Hash Table)

- Các thuật toán tìm kiếm đều dựa vào việc so sánh giá trị khoá (Key)
  - Phụ thuộc kích thước của tập các phần tử
  - Thời gian tìm kiếm không nhanh do phải thực hiện nhiều phép so sánh có thể không cần thiết (O(n), O(logn), …)

=> Có phương pháp lưu trữ nào cho phép thực hiện tìm kiếm với ***hiệu suất cao*** hơn không (độ phức tạp hằng số)?

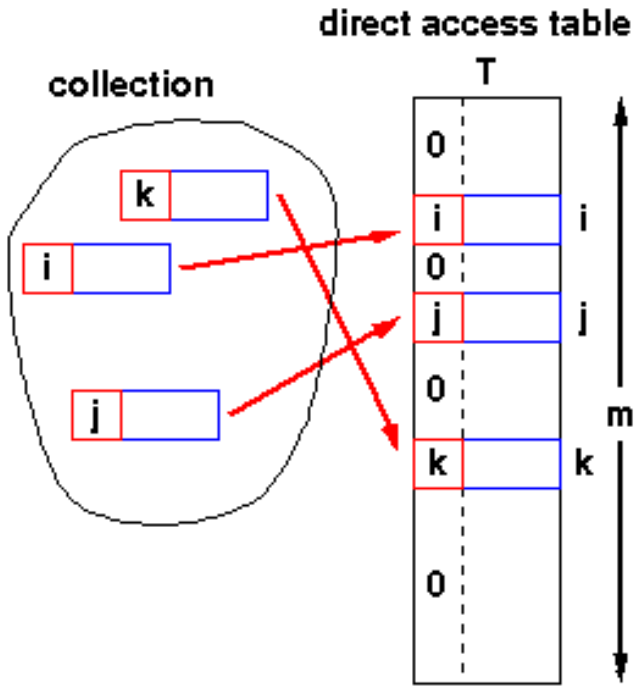
Tóm tắt Từ khoá

- **Bảng băm** (Hash Table) : là bảng cho phép thực hiện tìm kiếm các phần tử với thời gian O(1) thông qua một hàm tính địa chỉ gọi là hàm băm
- **Hàm băm** (Hash Function) : là hàm chuyển đổi giá trị khoá thành địa chỉ hay chỉ mục trong bảng băm
- **Sự đụng độ** (Collision) : xảy ra khi hàm băm 2 khoá khác nhau vào cùng 1 địa chỉ trên bảng băm

Bảng truy xuất trực tiếp

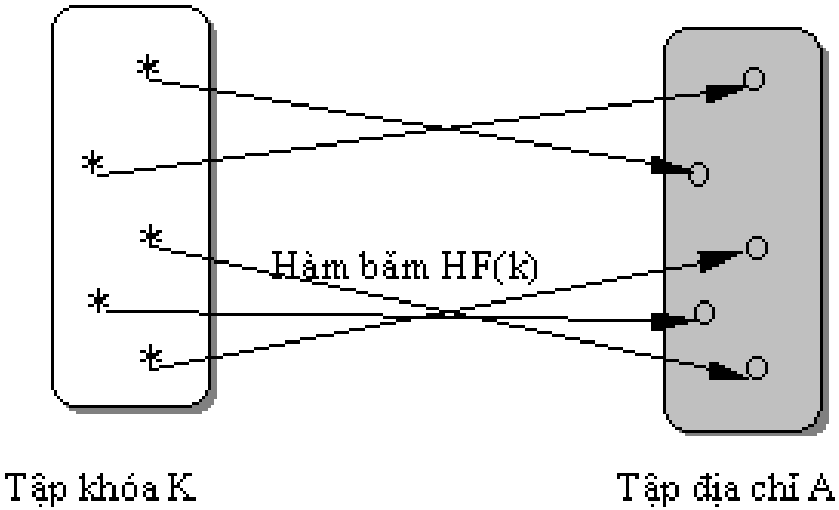
- Bảng gồm m phần tử được lưu trữ dưới dạng bảng chỉ mục
  - Phần tử có giá trị khoá k được lưu trữ tương ứng tại vị trí thứ k
  - Tìm kiếm bằng cách tra trong bảng chỉ mục
  - Thời gian tìm kiếm là O(1)

➔ **Đây là dạng bảng băm cơ bản**



Cấu trúc bảng băm

- K: tập các giá trị khoá (set of keys) cần lưu trữ
- A: tập các địa chỉ (set of addresses) trong bảng băm
- **HF(k): hàm băm dùng để ánh xạ một khoá k từ tập các khoá K thành một địa chỉ tương ứng trong tập các địa chỉ A**



Hình 1.2

Phân loại bảng băm

- ***Bảng băm đóng*** :
  - Số phần tử cố định
  - Mỗi khóa ứng với một địa chỉ
  - Không thể thực hiện các thao tác thêm, xóa trên bảng băm
  - thời gian truy xuất là hằng số
- ***Bảng băm mở*** :
  - Số phần tử không cố định
  - Một số khóa có thể có cùng địa chỉ
  - Có thể thực hiện các thao tác thêm, xóa phần tử
  - Thời gian truy xuất có thể bị suy giảm đôi chút

Hàm băm (Hash function)

- Là hàm biến đổi giá trị khoá (số, chuỗi…) thành địa chỉ, chỉ mục trong bảng băm

Ví dụ : hàm băm biến đổi khóa chuỗi thành 1 địa chỉ (số nguyên)

```
int hashfunc( char *s, int n )
{       int sum = 0;
        while( n-- ) sum = sum + *s++;
        return sum % 256;
}
```

- Tính địa chỉ của khoá "AB" : **hashfunc("AB",2) → 131**
- Tính địa chỉ của khoá "BA" : **hashfunc("BA",2) → 131**

→ **Khi hàm băm 2 khoá vào cùng 1 địa chỉ gọi là đụng độ (Collision)**

Hàm băm (Hash function)

- Tiêu chuẩn đánh giá hàm băm
  - Tính toán nhanh.
  - Các khoá được phân bố đều trong bảng.
  - Ít xảy ra đụng độ .

Phương pháp xây dựng hàm băm

- Hàm băm dạng bảng tra
- Hàm băm dùng phương pháp chia
- Hàm băm dùng phương pháp nhân

Phương pháp xây dựng hàm băm

- Hàm băm dạng bảng tra

Khoá	Địa chỉ	Khóa	Địa chỉ	Khóa	Địa chỉ	Khóa	Địa chỉ
a	0	h	7	o	14	v	21
b	1	i	8	p	15	w	22
c	2	j	9	q	16	x	23
d	3	k	10	r	17	y	24
e	4	l	11	s	18	z	25
f	5	m	12	t	19	/	/
g	6	n	13	u	20	/	/

Phương pháp xây dựng hàm băm

- **Hàm băm dùng phương pháp chia**
- Sử dụng số dư của phép chia để làm địa chỉ:

  $h(k) = k \bmod m$

  k là khoá, m là kích thước (số địa chỉ) của bảng.

  ➔ vấn đề chọn giá trị m

→ nên chọn m là nguyên tố

Phương pháp xây dựng hàm băm

- Ví dụ: Ta có tập khoá là các giá trị số gồm 3 chữ số, và vùng nhớ cho bảng địa chỉ có khoảng 100 mục, như vậy ta sẽ lấy hai số cuối của khoá để làm địa chỉ theo phép chia dư cho 100.

  Vd: 325 Mod 100 = 25, 125 Mod 100=25...

M=100	
Khoá	Địa chỉ
325	25
125	25
147	47

M=97 (nguyên tố)	
Khoá	Địa chỉ
325	34
125	28
147	50

Phương pháp xây dựng hàm băm

- **Hàm băm dùng phương pháp nhân**
- Sử dụng công thức:

$$h(k) = \lfloor m*(k*A \bmod 1) \rfloor$$

h(k) = floor(m (k A mod 1))

với k là khóa, m là kích thước bảng

A là hằng số: 0 < A < 1

Phương pháp xây dựng hàm băm

- *Vấn đề chọn m và A*
  - Ta thường chọn $m = 2^n$ hoặc $m = 10^n$
  - Theo Knuth: chọn A = 1/2(sqrt(5) -1) ≈ 0.618033987 được xem là tốt

Donald Ervin Knuth

http://www-cs-faculty.stanford.edu/~uno/

Phương pháp xây dựng hàm băm

- Ví dụ: Ta có tập khoá là các giá trị số gồm 3 chữ số, và vùng nhớ cho bảng địa chỉ có khoảng 100 mục, chọn hằng số A=0.61803

→ ***Tính địa chỉ cho khóa 325***

*h(325) = floor(100 (325*0.61803 mod 1))=86*

M=100, A=0.61803	
Khoá	**Địa chỉ**
325	86
125	25
147	85

M=100, A=0.52173	
Khoá	**Địa chỉ**
325	56
125	21
147	69

Các thao tác trên bảng băm

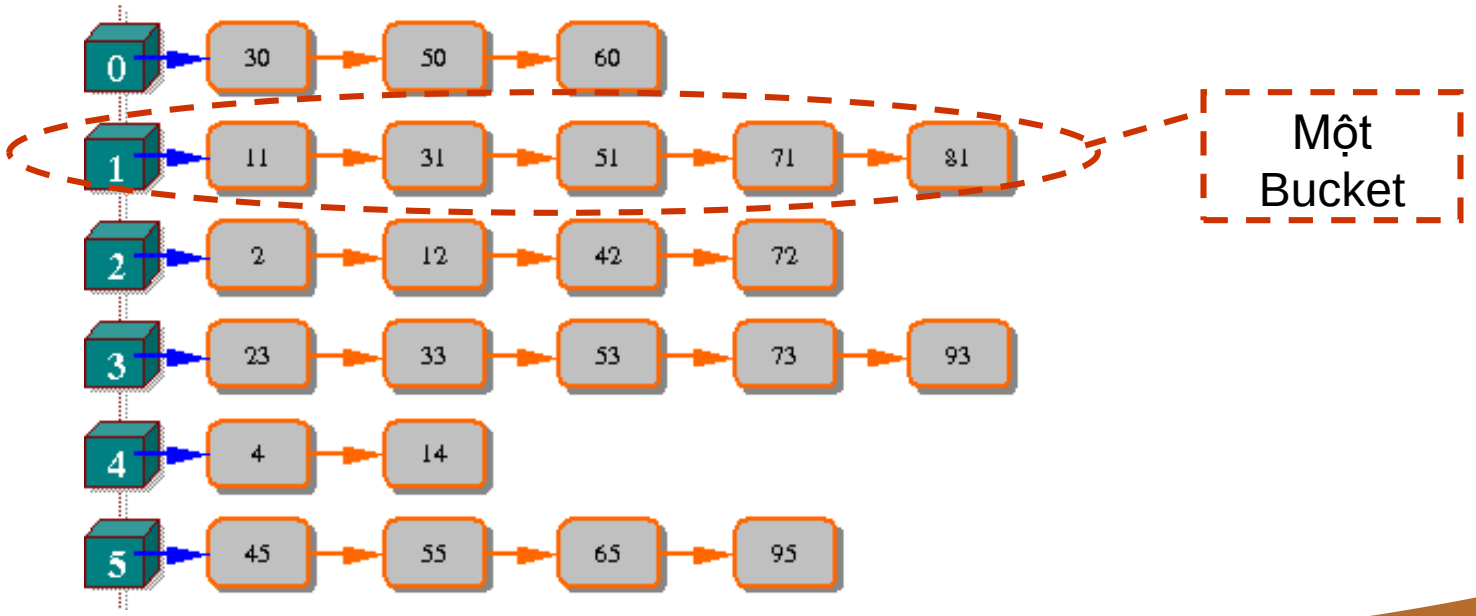
- Khởi tạo (*Initialize*)
- Kiểm tra rỗng (*Empty*)
- Lấy kích thước của bảng băm (*Size*)
- Tìm kiếm (*Search)*
- Thêm mới phần tử (*Insert*)
- Loại bỏ (*Remove*)
- Sao chép (*Copy*)
- Duyệt (*Traverse*)

Các phương pháp giải quyết đụng độ

- Phương pháp nối kết
- Phương pháp dò tuyến tính
- Phương pháp dò bậc hai
- Phương pháp dùng hàm băm kép

Các phương pháp giải quyết đụng độ

- **Phương pháp nối kết**
  - Các phần tử bị đụng độ được gom thành một danh sách liên kết (gọi là một bucket).



Cài đặt bảng băm phương pháp nối kết

- Khai báo cấu trúc bảng băm:

```
#define M 100
struct node
{   int key;
    struct node *next;
};
```

- Khai báo kiểu con trỏ chỉ nút

```
typedef struct nodes *nodeptr;
```

- Khai báo mảng bucket chứa M con trỏ đầu của M bucket

```
nodeptr bucket[M];
```

Cài đặt bảng băm phương pháp nối kết

- **Hàm băm**

```
int hashfunc (int key)
{   return (key % M); }
```

- **Phép toán khởi tạo (initbuckets)**

```
void initbuckets( )
{   int b;
    for (b=0;b<M;b++) bucket[b]=NULL;
}
```

- **Phép toán kiểm tra bucket rỗng (isemptybucket)**

```
int isemptybucket (int b)
{   return(bucket[b] ==NULL ?TRUE :FALSE);     }
```

Cài đặt bảng băm phương pháp nối kết

- **Phép toán kiểm tra bảng băm rỗng isempty:**

```
int isempty( )
{   int b;
    for (b=0;b<M;b++)
    if(bucket[b] !=NULL)  return(FALSE);
    return(TRUE);
}
```

- **Phép toán chèn phần tử có khóa k vào bảng băm:**

```
void insert(int k)
{   int b;
    b=hashfunc(k);
    place(b,k); //chen k vao danh sach lien ket
}
```

Cài đặt bảng băm phương pháp nối kết

- **Phép toán hủy mục có khóa k trong bảng băm**

```
void remove(int k)
{   int b;
    nodeptr q, p;
    b = hashfunc(k);  p = bucket[b];
    while(p!=NULL && p->key !=k)
    {      q=p;  p=p->next;  }
    if (p == NULL)
          printf("\n khong co nut co khoa %d" ,k);
    else if (p==bucket[b]) pop(b);
          else  delafter(q); //xoa nut
}
```

Cài đặt bảng băm phương pháp nối kết

- **Phép toán xóa bucket trong bảng băm**

```
void clearbucket (int b)
{   nodeptr p,q;
//q la nut truoc,p la nut sau
    q = NULL;
    p = bucket[b];
    while(p !=NULL)
    {      q = p;
           p=p->next;
           freenode(q);
    }
    bucket[b] = NULL; //khoi dong lai bucket b
}
```

Cài đặt bảng băm phương pháp nối kết

- **Phép toán xóa tất cả các phần tử trong bảng băm.**

```
void clear( )
{   int b;
    for (b=0;b<M;b++) clearbucket(b);
}
```

- **Phép toán duyệt các phần tử trong bucket b.**

```
void traversebucket (int b)
{   nodeptr p;  p=bucket[b];
    while (p!=NULL)
    {     printf("%5d", p->key);
          p= p->next;
    }
}
```

Cài đặt bảng băm phương pháp nối kết

- **Phép toán duyệt toàn bộ bảng băm:**

```
void traverse( )
{
    int b;
    for(b=0;b<M; b++)
    {
        printf("\nBucket thu %d:",b);
        traversebucket(b);
    }
}
```

Cài đặt bảng băm phương pháp nối kết

- **Phép toán tìm kiếm một phần tử trong bảng**

```
nodeptr search(int k)
{
    nodeptr p;
    int b;
    b = hashfunc (k);
    p = bucket[b];
    while(k>p->key && p!=NULL) p=p->next;
    if (p==NULL || k!=p->key)// khong tim thay
          return(NULL);
    else
          return(p);
}
```

Các phương pháp giải quyết đụng độ

- **Dò tuyến tính (Linear Probing Method):** là một phương pháp băm lại (Rehash), để chọn một địa chỉ kế tiếp trong bảng băm khi xảy ra đụng độ về địa chỉ, bằng cách cộng thêm một đơn vị

Các phương pháp giải quyết đụng độ

- Phương pháp dò tuyến tính
- Ý tưởng: Nếu vị trí hiện tại đã bị khóa khác chiếm, thử xét ô kế tiếp trong bảng:

```
linear_probing_insert(K)
    if (table is full) error
    probe = h(K)
    while (table[probe] occupied)
          probe = (probe + 1) mod M
    table[probe] = K
```

Phương pháp dò tuyến tính

- Xét dọc theo bảng cho đến khi tìm thấy khóa đang xét hoặc tìm thấy một ô trống.
- Ít tốn bộ nhớ hơn dùng danh sách kiên kết (chaining)
  - Không phải lưu các liên kết
- Nhưng chậm hơn dùng danh sách kiên kết.
  - Có thể phải duyệt dọc theo bảng trên con đường dài

Phương pháp dò tuyến tính

- Khó khăn:
  - Các phần tử bị đụng độ có xu hướng bị dồn cục
  - Kích thước bảng bị giới hạn
- Ví dụ
  - h(K) = K mod 13

Cài đặt bảng băm phương pháp dò tuyến tính

- Khai báo cấu trúc bảng băm:

```
#define NULLKEY -1
#define M 100
struct node
{   int key;
};
```

- Khai báo bảng băm

```
struct nodes hashtable[M];
```

- Khai báo biến số nút hiện có trong bảng

```
int sonut;
```

Cài đặt bảng băm phương pháp dò tuyến tính

- **Hàm băm**

```
int hashfunc (int key)
{   return (key % M); }
```

- **Phép toán khởi tạo (initbuckets)**

```
void initialize( )
{   int i;
    for(i=0;i<M;i++)
    hashtable[i].key=NULLKEY;
    N=0;  //so nut hien co khoi dong bang 0
}
```

Cài đặt bảng băm phương pháp dò tuyến tính

- **Phép toán kiểm tra bucket rỗng (isemptybucket)**

```
int empty( )
{
    return(N==0 ? TRUE:FALSE);
}
```

- **Phép toán kiểm tra bảng băm đầy isempty:**

```
int full( )
{
    return (N==M-1 ? TRUE: FALSE);
}
```

Lưu ý bảng băm đầy khi N=M-1, chúng ta nên dành ít nhất một phần tử trống trên bảng băm.

Cài đặt bảng băm phương pháp dò tuyến tính

- **Phép toán thêm khóa k vào bảng băm**

```
int insert(int k)
{    int i, j;
     if(full( ))
     {      printf("\n Bang bam bi day khong them nut
                    co khoa %d duoc",k);  return;
     }
     i=hashfunc(k);
     while(hashtable[i].key !=NULLKEY)
     {      //Bam lai (theo phuong phap do tuyen tinh)
            i ++;  if(i >M)   i= i-M;
     }
     hashtable[i].key=k;
     N=N+1;
     return(i);
}
```

Cài đặt bảng băm phương pháp dò tuyến tính

- **Phép toán tìm kiếm một phần tử trong bảng**

```
int search(int k)
{    int i;
     i=hashfunc(k);
     while(hashtable[i].key!=k && hashtable[i].key
                                        !=NULKEY)
     {//bam   lai   (theo   phuong   phap   do   tuyen   tinh:
     //fi(key)=f(key)+i) % M
            i=i+1;
            if(i>=M)
            i=i-M;
     }
     if(hashtable[i].key==k) //tim thay
            return(i);
     else //khong tim thay
            return(M);
}
```

Các phương pháp giải quyết đụng độ

- **Dò bậc hai (Quadratic Probing Method):** là một phương pháp băm lại để chọn một địa chỉ kế tiếp trong bảng băm khi xảy ra đụng độ về địa chỉ bằng cách cộng thêm i^2 đơn vị vào địa chỉ
- **Phương pháp băm kép (Double hashing Method):** là một phương pháp băm lại dùng cùng lúc hai hàm băm