Bài giảng môn học

Chương 04
Bảng Băm

ThS. Nguyen Minh Phuc
Software Technology Department - Faculty of Information Technology – Lac Hong University

Cấu trúc dữ liệu và giải thuật

Nội dung

- Bảng băm
- Định nghĩa hàm băm
- Một số phương pháp xây dựng hàm băm
- Các phương pháp giải quyết đụng độ

Yêu cầu

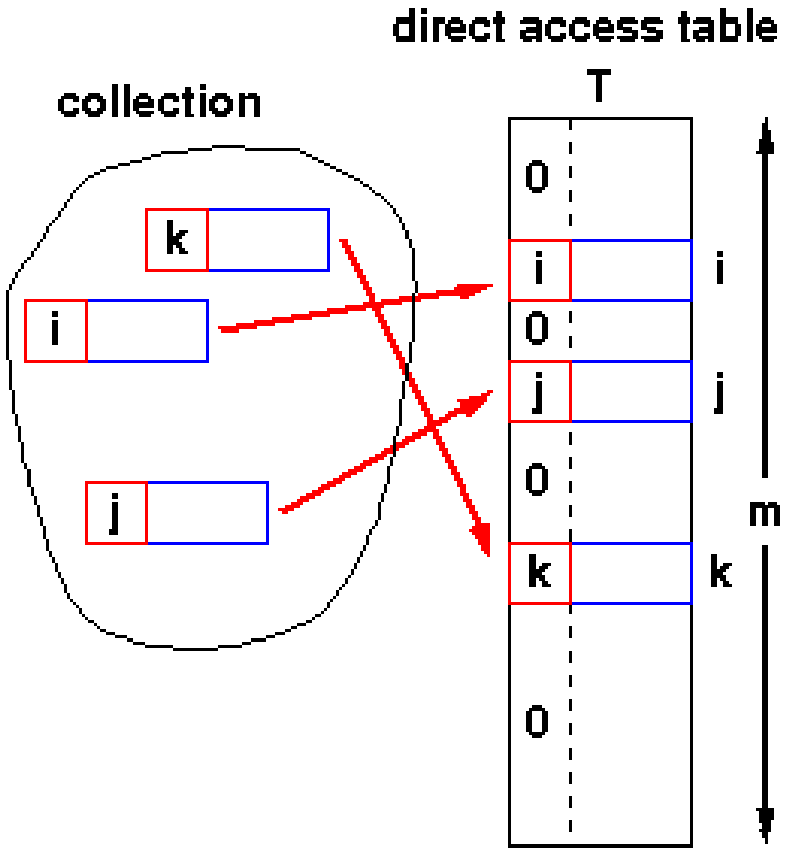
- Các thuật toán tìm kiếm đều dựa vào việc so sánh giá trị khoá (Key) của phần tử cần tìm với các giá trị khoá trong tập các phần tử, thao tác này Phụ thuộc kích thước của tập các phần tử
- Thời gian tìm kiếm không nhanh do phải thực hiện nhiều phép so sánh có thể không cần thiết (O(n), O(logn), …)
- => Có phương pháp lưu trữ nào cho phép thực hiện tìm kiếm với hiệu suất cao hơn không (độ phức tạp hằng số)?

Bảng băm

- Ví dụ: Giả sử ta có một tập phần tử gồm các giá trị khoá bất kỳ được tổ chức lưu trữ dưới dạng bảng chỉ mục m phần tử như sau gọi là bảng truy xuất trực tiếp (Direct access table).
- Phần tử có giá trị khoá k được lưu trữ tương ứng tại vị trí thứ k trong bảng chỉ mục
- Để tìm kiếm một phần tử nào đó ta sẽ dựa vào khoá của nó và tra trong bảng chỉ mục, nếu tại vị trí đó có phần tử thì chính là phần tử cần tìm, nếu không có phần tử nào có nghĩa là phần tử cần tìm không có trong bảng chỉ mục
- Thời gian tìm kiếm là hằng số O(1)

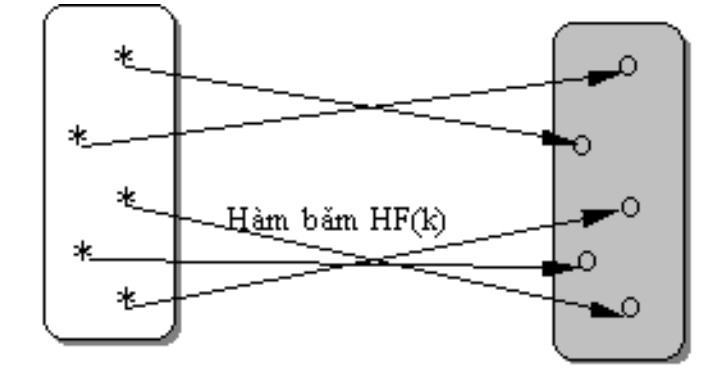
➔Đây là dạng bảng băm cơ bản

Minh họa



Mô tả dữ liệu

- Giả sử
  - K: tập các khoá (set of keys)
  - A: tập các địa chỉ (set of addresses).
  - HF(k): hàm băm dùng để ánh xạ một khoá k từ tập các khoá K thành một địa chỉ tương ứng trong tập A.



Hình 1.2

Các phép toán trên bảng băm

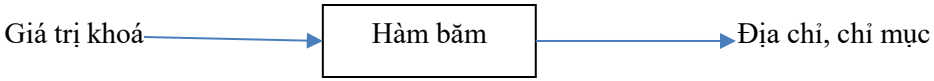
- Khởi tạo (*Initialize*)
- Kiểm tra rỗng (*Empty*)
- Lấy kích thước của bảng băm (*Size*)
- Tìm kiếm (*Search)*
- Thêm mới phần tử (*Insert*)
- Loại bỏ (*Remove*)
- Sao chép (*Copy*)
- **Duyệt (*Traverse*)**

Phân loại bảng băm

- *Bảng băm đóng* : mỗi khóa ứng với một địa chỉ, thời gian truy xuất là hằng số
- *Bảng băm mở* : một số khóa có cùng địa chỉ, lúc này mỗi mục địa chỉ sẽ là một danh sách liên kết các phần tử có cùng địa chỉ, thời gian truy xuất có thể bị suy giảm đôi chút.

Định nghĩa hàm băm

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



Ví dụ : hàm băm biến đổi khoá dạng chuỗi gồm n kí tự 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 thì gọi là đụng độ (Collision)

Định nghĩa hàm băm

- 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 .

Một số phương pháp xây dựng hàm băm

1. Hàm băm dạng bảng tra.
2. Hàm băm sử dụng phương pháp chia
3. Hàm băm sử dụng phương pháp nhân
4. Dùng hàm băm phổ quát

Hàm băm dạng bảng tra

- Hàm băm có thể tổ chức ở dạng bảng tra (còn gọi là bảng truy xuất) hoặc thông dụng nhất là ở dạng công thức.
- Ví dụ sau đây là bảng tra với khóa là bộ chữ cái, bảng băm có 26 địa chỉ từ 0 đến 25. Khóa a ứng với địa chỉ 0, khoá b ứng với địa chỉ 1,… , z ứng với địa chỉ 25.

Ví dụ

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	/	/

Hàm băm sử dụng phương pháp chia

- Dùng số dư:
  - $h(k) = k \bmod m$
  - k là khoá, m là kích thước của bảng.

➔ vấn đề chọn giá trị m

- Nếu chọn m= 2^n thông thường không tốt vì h(k) = k mod 2^n sẽ chọn cùng n bits thấp của k → nên chọn m là nguyên tố (tốt) gần với 2^n
- 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 lấy dư cho 100 : chẳng hạn 325 Mod 100 = 25
- Tuy nhiên ta nhận thấy nếu hàm băm dùng công thức như trên thì địa chỉ của khoá tính được chỉ căn cứ và 2 ký số cuối. Vì thế, để hàm băm có thể tính địa chỉ khoá một cách "ngẫu nhiên" hơn ta nên chọn m=97 thay vì 100

Ví dụ

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

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

Hàm băm sử dụng phương pháp nhân

- Sử dụng công thức
  - $h(k)$ = floor(m (k A mod 1))
  - k là khóa, m là kích thước bảng, A là hằng số: $0 < A < 1$
- *Chọn m và A*
  - Ta thường chọn $m = 2^n$
  - Theo Knuth chọn A = 1/2(sqrt(5) -1) ≈ 0.618033987 được xem là tốt.

Ví dụ

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

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

Dùng hàm băm phổ quát

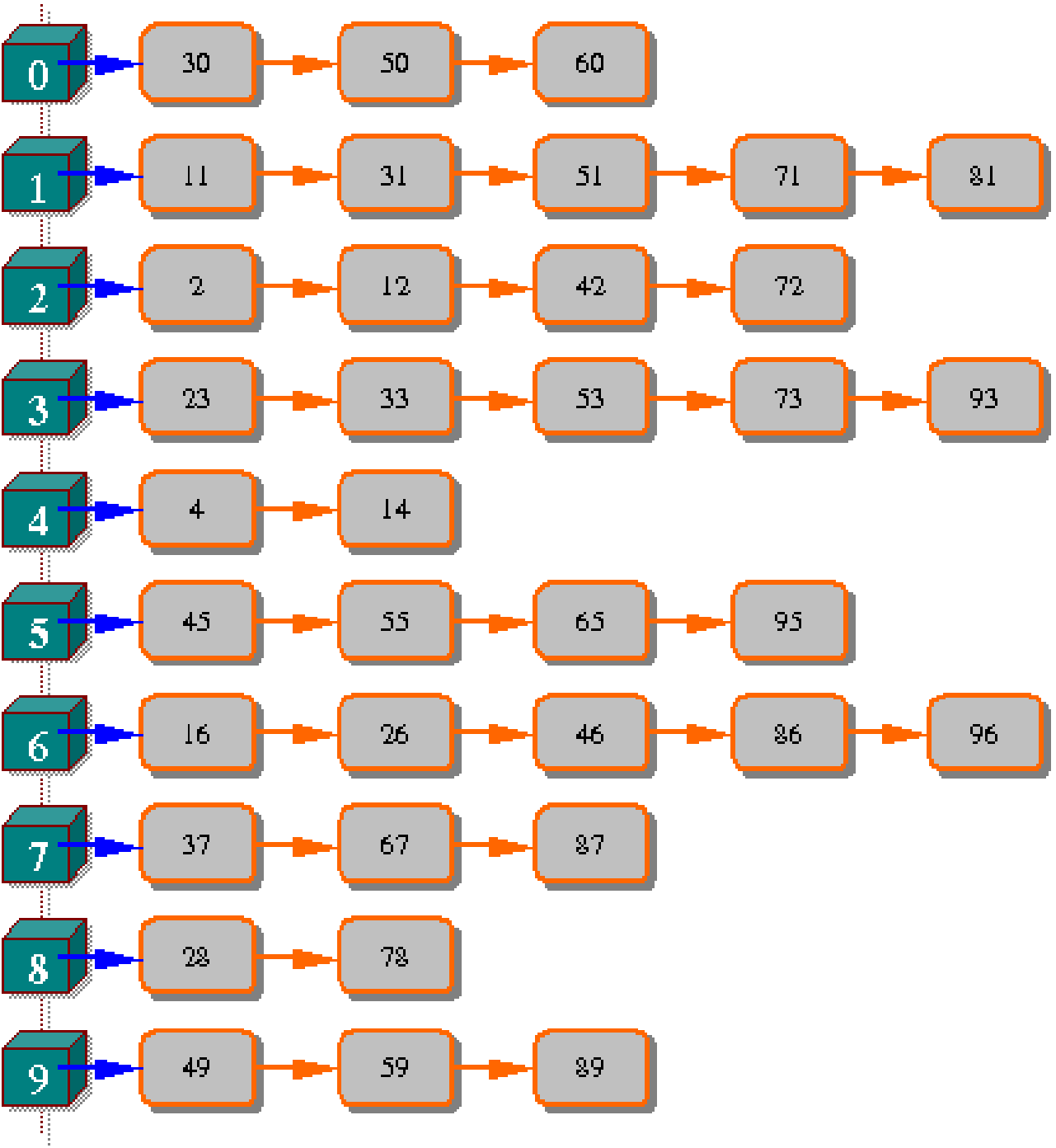
- Một tập các hàm băm H là phổ quát (*universal*) nếu $\forall h \in H$ và 2 khoá k, l ta có xác suất: $Pr\{h(k) = h(l)\} <= 1/m$,· với m là kích thước bảng
- Để xác suất xảy ra đụng độ thấp, khởi tạo một tập các hàm băm H phổ quát và từ đó h được chọn ngẫu nhiên.

Một số phương pháp giải quyết sự đụng độ

- Người ta giải quyết sự đụng độ theo hai phương pháp: *phương pháp nối kết* và *phương pháp băm lại*.

Giải quyết sự đụng độ bằng phương pháp nối kết (Chaining Method):

- Các phần tử bị băm vào cùng địa chỉ (các phần tử bị đụng độ) được gom thành một danh sách liên kết (gọi là một bucket).


0
30
50
60
1
11
31
51
71
81
2
2
12
42
72
3
23
33
53
73
93
4
4
14
5
45
55
65
95
6
16
26
46
86
96
7
37
67
87
8
28
78
9
49
59
89

Cài đặt bảng băm dùng phương pháp nối kết trực tiếp

- *Khai báo cấu trúc bảng băm:*

```
#define M   100
struct nodes
{
      int key;
      struct nodes *next
};
//khai bao kieu con tro chi nut
typedef struct nodes *NODEPTR;
//khai bao mang bucket chua M con tro dau cua M bucket
NODEPTR bucket[M];
```

Các phép toán

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

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

Các phép toán

- Phép toán initbuckets:

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

Các phép toán

- **Phép toán emmptybucket:**

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

Các phép toán

- **Phép toán emmpty:**

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

Các phép toán

- **Phép toán insert:**

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

Các phép toán

- **Phép toán remove:**

```
void remove ( int k)
{
          int b;
          NODEPTR q, p;
          b = hashfunc(k);
          p = hashbucket(k);
          q=p;
          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);
//Tac vu pop cua danh sach lien ket
else
          delafter(q);
/*tac vu delafter cua danh sach lien ket*/
}
```

Các phép toán

- **Phép toán clearbucket:** Xóa tất cả các phần tử trong bucket b.

```
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 butket b
}
```

Các phép toán

- **Phép toán clear:** Xóa tất cả các phần tử trong bảng băm.

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

Các phép toán

- **Phép toán traversebucket:** Duyệt các phần tử trong bucket b.

```
void traversebucket (int b)
{
     NODEPTR p;
     p= bucket[b];
     while (p !=NULL)
     {
          printf("%3d", p->key);
          p= p->next;
     }
}
```

Các phép toán

- **Phép toán traverse:** Duyệt toàn bộ bảng băm.

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

Các phép toán

- **Phép toán search:** Tìm kiếm một phần tử trong bảng băm,nếu không tìm thấy hàm này trả về giá trị NULL,nếu tìm thấy hàm này trả về con trỏ chỉ tìm phần tử tìm thấy.

```
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//tim thay
        return(p);
}
```

Giải quyết sự đụng độ bằng phương pháp băm lại

- Phương pháp dò tuyến tính (Linear Probe)
  - 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.
  - $h_i(key)=(h(key)+i)\ \%M$ với h(key) là hàm băm chính của bảng băm

Cài đặt bảng băm dùng phương pháp dò tuyến tính

- *Khai báo cấu trúc bảng băm:*

```
#define NULLKEY -1
#define M 100
/*
M la so nut co tren bang bam,du de chua cac nut nhap vao bang bam
*/
//khai bao cau truc mot nnut cua bang bam
struct node
{
        int key; //khoa cua nut tren bang bam
};
//Khai bao bang bam co M nut
struct node hashtable[M];
int NODEPTR;
/*bien toan cuc chi so nut hien co tren bang bam*/
```

Các tác vụ

- Hàm băm:

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

-

- Phép toán khởi tạo (initialize): Khởi tạo bảng băm. Gán biến toàn cục N=0.

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

- Phép toán kiểm tra trống (empty): Kiểm tra bảng băm có trống hay không.

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

- **Phép toán kiểm tra đầy (full):** Kiểm tra bảng băm đã đầy chưa.

```
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.

- **Phép toán search:**

```
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);
}
```

- **Phép toán insert:**
- Thêm phần tử có khoá 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);
}
```

Bảng băm với phương pháp dò bậc hai (Quadratic Probing Method)

- 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:
- $h_i(key)=(h(key) + i2) \% M$
- với h(key) là hàm băm chính của bảng băm.
- 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.

Cài đặt bảng băm dùng phương pháp dò bậc hai

- Khai báo cấu trúc bảng băm:

```
#define NULLKEY –1
#define M 101
/* M la so nut co tren bang bam,du de chua cac nut nhap vao bang bam,chon M la so nguyen to
*/
//Khai bao nut cua bang bam
struct node
{
int key; //Khoa cua nut tren bang bam
};
//Khai bao bang bam co M nut
struct node hashtable[M];
int N;
//Bien toan cuc chi so nut hien co tren bang bam
```

- Hàm băm: Giả sử chúng ta chọn hàm băm dạng%: f(key)=key %10.

```
int hashfunc(int key)
{
        return(key% 10);
}
```

- Phép toán initialize Khởi động hàm băm. Gán biến toàn cục N=0.

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

- **Phép toán empty:** Kiểm tra bảng băm có rỗng không

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

- Phép toán full:
- Kiểm tra bảng băm đã đầy chưa .

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

- Lưu ý bảng băm đầy khi N=M-1 chúng ta nên chừa ít nhất một phần tử trong trên bảng băm!

- Phép toán search:
- Tìm phần tử có khóa k trên bảng băm,nếu không tìm thấy hàm này trả về trị M, nếu tìm thấy hàm này trả về địa chỉ tìm thấy.

```
int INTSERT(int k)
{
        int i, d;
        i = hashfuns(k);
        d = 1;
        while(hashtable[i].key!=k&&hashtable[i].key !=NULLKEY)
        {
                //Bam lai (theo phuong phap bac hai)
                i = (i+d) % M;
                d = d*2;
        }
        hashtable[i].key =k;
        N = N+1;
        return(i);
}
```

Cấu trúc dữ liệu và giải thuật