Cấu trúc dữ liệu

Data Structure

Ts. Nguyễn Đức Thuần
BM Hệ thống Thông Tin

Chương VII

BẢNG BĂM (HASH TABLE)

Bảng băm

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

Giới thiệu

- Phép băm được đề xuất và hiện thực trên máy tính từ những năm 50 của thế kỷ 20.
- Ý tưởng: biến đổi giá trị khóa thành một số (xử lý băm) và sử dụng số này để đánh chỉ số cho bảng dữ liệu: Bảng chỉ mục (bảng băm)

Các phép toán trên bảng băm sẽ giúp hạn chế số lần so sánh

☞ giảm thiểu được thời gian truy xuất.

(*Độ phức tạp của các phép toán trên bảng băm thường có bậc là 0(1) và không phụ thuộc vào kích thước của bảng băm*)

Ví dụ:

Phân 100 học sinh được đánh số thự tự lần lượt vào 10 phòng thi. Tìm phòng thi của học sinh thứ i?

Bảng băm

- Bảng băm (hash table)
- *Định nghĩa*: HashTable là một kiểu dữ liệu danh sách dùng để lưu dữ liệu theo một từ khóa và giá trị của nó.

Address	Key	Values
001	a	Ant
002	b	Book
…	…	….
025	y	Yatch
026	z	Zebra

- Phân loại
- *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

Bảng băm

Các khái niệm chính trên cấu trúc 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*)

Thông thường bảng băm được sử dụng khi cần xử lý các bài toán có dữ liệu lớn và được lưu trữ ở bộ nhớ ngoài.

CÁC HÀM C PHỤC VỤ CHO BẢNG BĂM

a. Khai báo cấu trúc bảng băm:

C Code:

```c
#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*/
```

b. Các tác vụ:

Hàm băm:

C Code:

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

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

C Code:

```c
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.

C Code:

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

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

C Code:

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

CÁC HÀM C PHỤC VỤ CHO BẢNG BĂM

Phép toán search:

C Code:

```c
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:

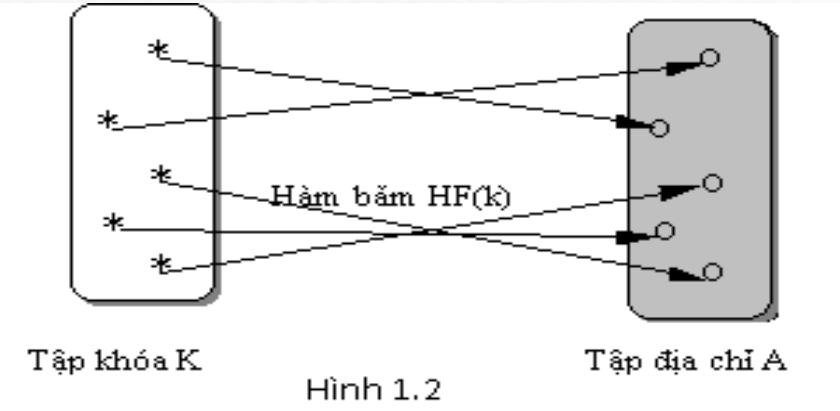
Thêm phần tử có khoá k vào bảng băm.

C Code:

```c
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);
}
```

Hàm băm

- 1. HÀM BĂM (Hash Function)
- *Định nghĩa*: Hàm băm là ánh xạ từ tập giá trị khóa vào một tập các địa chỉ (của bảng băm).



Hình 1.2

- *Ví dụ* : hàm băm biến đổi khoa dạng chuoi gom n ki tự thanh 1 dịa chi (so 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**

Hàm băm

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

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**

a. 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.

Hàm băm

Ví dụ: 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.

Khóa	Địa chỉ	Khóa	Địa chỉ	Khóa	Địa chỉ	Khóa	Địa chỉ
a	1	h	8	o	15	v	22
b	2	i	9	p	16	x	23
c	3	j	10	q	17	y	24
d	4	k	11	r	18	w	25
e	5	l	12	s	19	z	26
f	6	m	13	t	20		
g	7	n	14	u	21		

Hàm băm dạng bảng tra được tổ chức dưới dạng danh sách kề.

Hàm băm

- *b. Hàm băm sử dụng phương pháp chia*

  Dùng dư số: $h(k) = k \bmod m$

  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 $h(k) = k \bmod 2^n$ 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

Có nhiều số có cùng địa chỉ. Vì thế, để hàm băm có thể tính địa chỉ khoá ít trùng lắp hơn chọn m=97 thay vì m=100

(Ví dụ: 325 mod 100 = 25; 125 mod 100 = 25

325 mod 97 = 34; 125 mod 97 = 28

Hàm băm

- *c. Hàm băm sử dụng phương pháp nhân*

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

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: k=123456; m=10000

H(k)= ⌊10000 (123456* 0.6180339887 mod 1) ⌋

H(k)= ⌊10000 (76300.0041089472 mod 1) ⌋

H(k)= ⌊10000 (0.0041089472) ⌋

H(k)=41

Hàm băm

d. *Hàm băm phổ quát (universal hashing function)*

Định nghĩa: Cho H là một tập hợp hữu hạn các hàm băm: ánh xạ các khóa k từ tập khóa U vào miền giá trị {0,1,2,…, m-1}. Tập H là phổ quát nếu:

$\forall f \in H$ và 2 khoá phân biệt k1,k2 ta có xác suất: Pr{*f(k1)* = *f(k2)*} <= 1/*m.*

Ứng dụng:

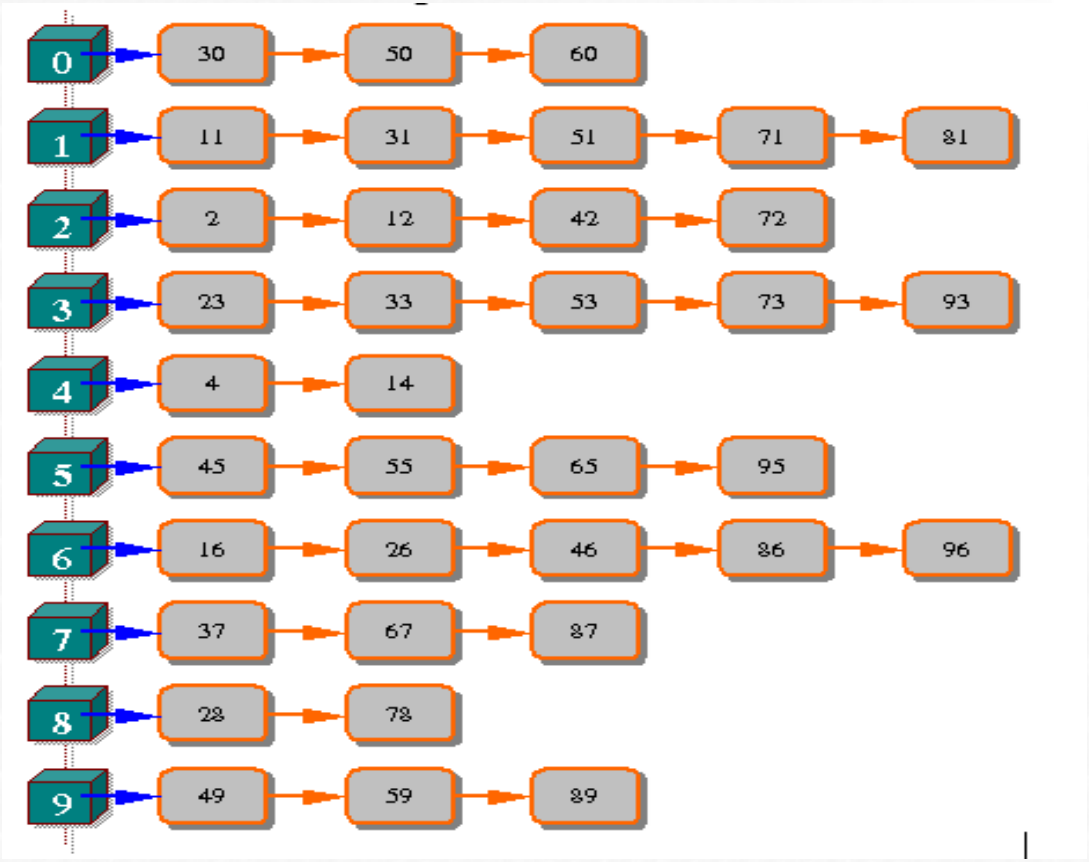
- Khởi tạo một tập các hàm băm H phổ quát
- Chọn $h \in H$ ngẫu nhiên.

Giải quyết đụng độ

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

***1. 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).*



Giải quyết đụng độ

- *a. 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];
```

Giải quyết đụng độ

- Các phép toán :

- Tính giá trị hàm băm:Giả sử chúng ta chọn hàm băm dạng %: h(key)=key % M.
- Phép toán initbuckets: khởi tạo các bucket bằng Null.
- Phép toán emmptybucket(b): kiểm tra bucket b có bị rỗng không?
- Phép toán emmpty: Kiểm tra bảng băm có rỗng không?
- Phép toán insert: Thêm phần tử có khóa k vào bảng băm.
    + i=h(k)
    + ktra bucket [i]: neu rong =>cc o nho cho bucket, gan khoa k
      them phan tu co khoa k vao ds theo thu tu tang dan.
- Phép toán remove: Xóa phần tử có khóa k trong bảng băm.
- Phép toán clear: Xóa tất cả các phần tử trong bảng băm.
- Phép toán traversebucket: Xử lý tất cả các phần tử trong bucket b.
- Phép toán traverse: Xử lý tất cả các phần tử trong bảng băm.
- Phép toán search: Tìm kiếm một phần tử trong bảng băm, nếu không tìm thấy hàm này trả về hàm NULL, nếu tìm thấy hàm này trả về địa chỉ của phần tử có khóa k.

ĐỊA CHỈ CÁC HÀM C PHỤC VỤ CHO BẢNG BĂM

- http://diendan.congdongcviet.com/threads/t3289::ly-thuyet-tim-kiem-bang-bam-hash-table.cpp

a. Khai báo cấu trúc bảng băm:
#

C Code:

```
1. define M   100
2. struct nodes
3. {
4. int key;
5. struct nodes *next
6. };
```

C Code:

```
1. //khai bao kieu con tro chi nut
2. typedef struct nodes *NODEPTR
3. /*
4. khai bao mang bucket chua M con tro dau cua M bucket
5. */
6. NODEPTR bucket[M];
```

b.Các phép toán:
Hàm băm
Giả sử chúng ta chọn hàm băm dạng %: f(key)=key % M.

C Code:

```
1. int hashfunc (int key)
2. {
3. return (key % M);
4. }
```

Phép toán initbuckets:

C Code:

```
1. void initbuckets( )
2. {
3. int b;
4. for (b=0;b<M;b++);
5. bucket[b]=NULL;
6. }
```

Phép toán emmptybucket:

C Code:

```
1. int emptybucket (int b)
2. {
3. return(bucket[b] ==NULL ?TRUE :FALSE);
4. }
```

Phép toán emmpty:

C Code:

```
1. int empty( )
2. {
3. int b;
4. for (b = 0;b<M;b++)
5. if(bucket[b] !=NULL)  return(FALSE);
6. return(TRUE);
7. }
```

CÁC HÀM C PHỤC VỤ CHO BẢNG BĂM

Phép toán remove:

C Code:

```c
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*/
}
```

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

C Code:

```c
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
}
```

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

C Code:

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

CÁC HÀM C PHỤC VỤ CHO BẢNG BĂM

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

C Code:

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

Phép toán traverse:
Duyệt toàn bộ bảng băm.

C Code:

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

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

C Code:

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

Giải quyết đụng độ

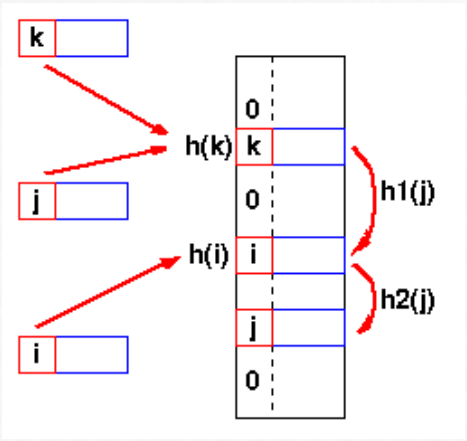
2. Giải quyết sự đụng độ bằng phương pháp băm lại: (Rehash Method)

- **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



Giải quyết đụng độ

- **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:

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

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.

CÁC HÀM C PHỤC VỤ CHO BẢNG BĂM

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

C Code:

```
1. int search(int k)
2. {
3. int i, d;
4. i = hashfuns(k);
5. d = 1;
6. while(hashtable[i].key!=k&&hashtable[i].key !=NULLKEY)
7. {
8. //Bam lai (theo phuong phap bac hai)
9. i = (i+d) % M;
10. d = d*2;
11. }
12. hashtable[i].key =k;
13. N = N+1;
14. return(i);
15. }
```

Giải quyết đụng độ

- **Phương pháp băm kép (Double hashing Method):** là một phương pháp băm lại dùng cùng lúc hai hàm băm

Ý tưởng:

- Dùng hai hàm băm h1(key) và h2(key)

- *Khi bổ sung 1 khóa* key: i=h1(key) và j=h2(key) sẽ xác định địa chỉ i và j trong khoảng từ 0 đến M-1:Khi thêm phần tử có khoá key vào bảng băm

  · 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 h1 sẽ xét địa chỉ mới i+j, nếu lại bị xung đột thì hàm băm lại lần 2 h2 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.

Giải quyết đụng độ

- *Khi tìm kiếm một phần tử* có khoá key xác định: i=h1(key) và j=h2(key)
- 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).

  Nếu đã dò đến cuối bảng thì trở về dò lại từ đầu bảng.

Cám ơn đã theo dõi