CHƯƠNG 4 - BẢNG BĂM (HASH TABLE)

Phép băm được đề xuất và hiện thực trên máy tính từ những năm 50 của thế kỷ 20. Nó 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. Chương này sẽ khảo sát một cấu trúc dữ liệu mới được gọi là bảng băm(hash table). 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 pháp toán trên bảng băm thường có bậc là 0(1) và không phụ thuộc vào kích thước của bảng băm.

Chương này sẽ giới thiệu các chủ đề và các phép toán chính thường dùng trên cấu trúc bảng băm:

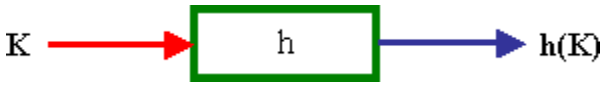
- Phép băm hay hàm băm (hash function)
- Tập khoá của các phần tử trên bảng băm
- Tập địa chỉ trên bảng băm
- Phép toán thêm phần tử vào bảng băm
- Phép toán xoá một phần tử trên bảng băm
- Phép toán tìm kiếm trên bảng băm

Thông thường bảng băm được sử dụng khi cần giải quyết những bài toán có các cấu trúc dữ liệu lớn và được lưu trữ ở bộ nhớ ngoài.

1. PHÉP BĂM (HASH FUNCTION)

Định nghĩa:

Trong hầu hết các ứng dụng, khoá được dùng như một phương thức để truy xuất dữ liệu một cách gián tiếp. Hàm được dùng để ánh xạ một khoá vào một dãy các số nguyên và dùng các giá trị nguyên này để truy xuất dữ liệu được gọi là hàm băm (hình 1)



Hình 1

Như vậy, hàm băm là hàm biến đổi khóa của phần tử thành địa chỉ trên bảng băm.

Khóa có thể là dạng số hay số dạng chuỗi.

- *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.
- *Giải quyết vấn đề băm với các khoá không phải là số nguyên:*
  - Tìm cách biến đổi khoá thành số nguyên
    - Ví dụ loại bỏ dấu '-' trong mã số 9635-8904 đưa về số nguyên 96358904

- Đối với chuỗi, sử dụng giá trị các kí tự trong bảng mã ASCCI
  - Sau đó sử dụng các hàm băm chuẩn trên số nguyên.

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
- $m = 2^n$ (không tốt)
- nếu chọn m= 2^n thông thường không tốt h(k) = k mod 2^n sẽ chọn cùng n bits cuối 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 $m = 4093$

Hàm Băm sử dụng Phương pháp nhân

- Sử dụng
  - $h(k) = m\ (k\,A \bmod 1)$
  - k là khóa, m là kích thước bảng, A là hằng số: $0 < A < 1$
- *Chọn m và A*
  - M thường chọn $m = 2p$
  - Sự tối ưu trong việc chọn A phụ thuộc vào đặc trưng của dữ liệu.
  - Theo Knuth chọn $A = 1/2(\sqrt{5} - 1) \approx 0.618033987$ được xem là tốt.

Phép băm phổ quát

- Việc chọn hàm băm không tốt có thể dẫn đến xác suất đụng độ lớn.
- Giải pháp:
  - Lựa chọn hàm băm h ngẫu nhiên.
  - Chọn hàm băm độc lập với khóa.
  - Khởi tạo một tập các hàm băm H phổ quát và từ đó h được chọn ngẫu nhiên.
- Một tập các hàm băm H là phổ quát (*universal*) nếu với mọi $\forall$ f, k $\in$ H và 2 khoá k, l ta có xác suất: $\Pr\{f(k) = f(l)\} <= 1/m$

Ví dụ: Giả sử nếu khoá là một số nguyên, dương và HK(key) là một số nguyên với một digit từ 0..9, Thế thì, hàm băm sẽ dùng toán tử modulo-10 để trả về giá trị tương ứng của một khoá. Chẳng hạn: nếu khoá=49 thì HF(49)=9.

Một cách tổng quát, với một hàm băm, nhiều khoá khác nhau có thể cho cùng một giá trị băm. Trong tình huống này xảy ra sự xung đột (collision) và cần thiết phải giải quyết sự đụng độ này. Một trong những phương pháp giải quyết sự xung đột với thời gian nhanh là sử dụng các cấu trúc danh sách đặc, hay danh sách kề có kích thước cố định. (xem phần 4)

Các cấu trúc bảng băm đơn giản, thường được cài đặt bằng các danh sách kề. Do vậy, để truy xuất một phần tử trên các bảng băm thuộc loại này, chỉ cần hai khóa tương ứng với hàng thứ i và cột thứ j để định vị một phần tử trên bảng.

Bảng băm chữ nhật (m hàng, n cột):

Mỗi phần tử trên bảng chữ nhật tương ứng với hai khóa tương ứng hàng thứ i và cột thứ j, địa chỉ phần tử này trên danh sách kề được xác định qua hàm băm:

0 ----------> j

0 | | | | V i

0	1	2	...	n
1		x		
2				
...				
...				
m				

Hình 1.2. Bảng băm chữ nhật

0	1	2	3	...	n-1	n	n+1	n+2	...	m x n

Danh sách kề mô tả bảng băm hình chữ nhật

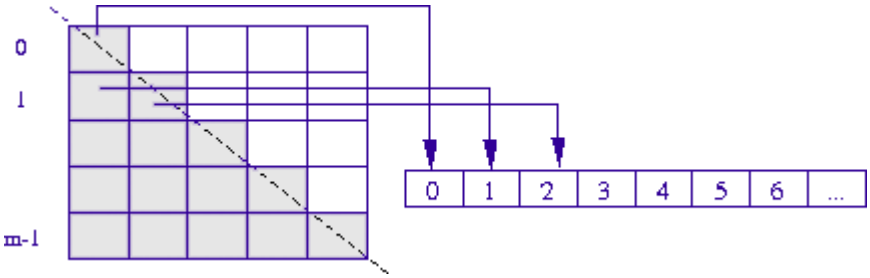
bảng băm: phần tử x thuộc hàng 2 cột 3 - f(1,2) = n + 3

Tổng quát, phần tử thuộc hàng i, cột j được cho bởi công thức:

f(i,j) =ni + j (n là số cột của bảng chữ nhật)

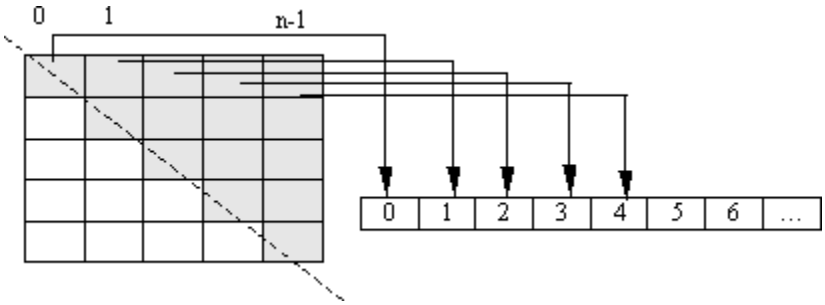
Bảng băm tam giác dưới (m hàng) và bảng băm tam giác trên (n cột):

Hình sau là bảng tam giác dưới m hàng



Hình 1.3.a Bảng băm tam giác dưới m hàng

Và bảng băm tam giác trên n cột



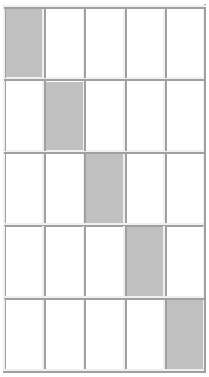
Hình 1.3.b Bảng băm tam giác trên n cột

Mỗi phần tử trên bảng tam giác dưới tương ứng với hai khóa hàng i, cột j(i>=j), địa chỉ phần tử này trên danh sách kề được xác định qua hàm băm:

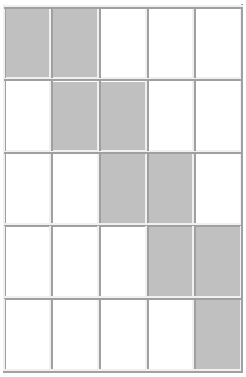
f(i,j)=i(i+1)/2 + j

Bảng băm đường chéo (n cột):

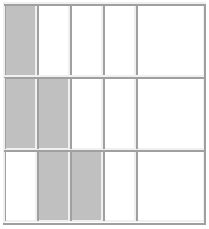
Hình sau là các dạng bảng đường chéo n cột, hãy xác định hàm băm cho các bảng đường chéo này.

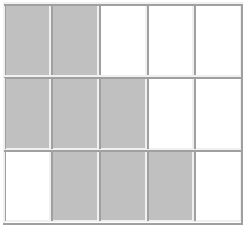


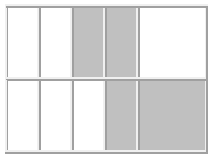
i = j



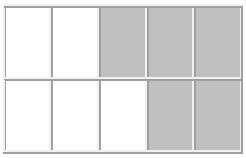
i = j hay i = j-1







i = j hay i = j+1



i = j hay i = j±1

Hình 1.4. Các bảng băm đường chéo

Như đã giới thiệu ở phần trên, với mỗi bảng băm đơn giản chúng ta cần xây dựng một hàm băm để truy xuất dữ liệu lưu trữ trong các phần tử trên bảng băm. Hàm băm thường có dạng công thức tổng quát **HF(key)** hay **f(khoá)** hoặc được tổ chức ở dạng bảng tra gọi là bảng truy xuất (access table).

2. BẢNG BĂM ADT (HASH TABLE - ADT)

Phần này sẽ trình bày các vấn đề chính:

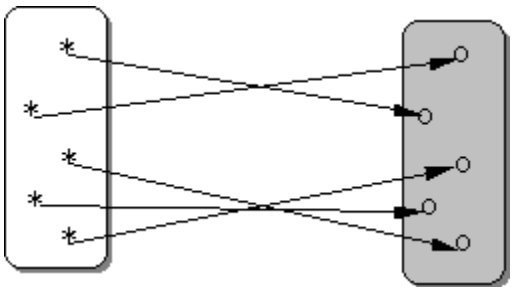
- Mô tả cấu trúc bảng băm tổng quát (thông qua hàm băm, tập khóa, tập địa chỉ…)
- Các phép toán trên bảng băm như thêm phần tử (insert), loại bỏ (remove), tìm kiếm (search), …

Bảng băm ADT:

a. Mô tả dữ liệu

Giả sử

- K: tập các khoá (set of keys)
- M: tập các dịa 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 M.



Tập khóa K Hàm băm Tập địa chỉ M

b. Các phép toán trên bảng băm

- Khởi tạo (*Initialize*): Khỏi tạo bảng băm, cấp phát vùng nhớ hay qui định số phần tử (kích thước) của bảng băm
- Kiểm tra rỗng (*Empty*): kiểm tra bảng băm có rỗng hay không?
- Lấy kích thước của bảng băm (*Size*): Cho biết số phần tử hiện có trong bảng băm
- Tìm kiếm (*Search*): Tìm kiếm một phần tử trong bảng băm theo khoá k chỉ định trước.

- Thêm mới phần tử (*Insert*): Thêm một phần tử vào bảng băm. Sau khi thêm số phần tử hiện có của bảng băm tăng thêm một đơn vị.
- Loại bỏ (*Remove*): Loại bỏ một phần tử ra khỏi bảng băm, và số phần tử sẽ giảm đi một.
- Sao chép (*Copy*): Tạo một bảng băm mới tử một bảng băm cũ đã có.
- Duyệt (*Traverse*): duyệt bảng băm theo thứ tự địa chỉ từ nhỏ đến lớn.

Các Bảng băm thông dụng:

Với mỗi loại bảng băm cần thiết phải xác định tập khóa K, xác định tập địa chỉ M và xây dựng hàm băm HF cho phù hợp.

Mặt khác, khi xây dựng hàm băm cũng cần thiết phải tìm kiếm các giải pháp để giải quyết sự xung đột, nghĩa là giảm thiểu sự ánh xạ của nhiều khoá khác nhau vào cùng một địa chỉ (ánh xạ nhiều-một).

Bảng băm với phương pháp nối kết trực tiếp: mỗi địa chỉ của bảng băm(gọi là một bucket) tương ứng một danh sách liên kết.

Các phần tử bị xung đột được nối kết với nhau trên một danh sách liên kết.

Bảng băm với phương pháp nối kết hợp nhất: bảng băm loại này được cài đặt bằng danh sách kề, mỗi phần tử có hai trường: trường *key* chứa khóa của phần tử và trường *next* chỉ phần tử kế bị xung đột. Các phần tử bị xung đột được nối kết nhau qua trường nối kết next.

Bảng băm với phương pháp dò tuyến tính: ví dụ khi thêm phần tử vào bảng băm loại này nếu băm lần đầu bị xung đột thì lần lượt dò địa chỉ kế.. cho đến khi gặp địa chỉ trống đầu tiên thì thêm phần tử vào địa chỉ này.

Bảng băm với phương pháp dò bậc hai: ví dụ khi thêm phần tử vào bảng băm loại này, nếu băm lần đầu bị xung đột thì lần lượt dò đến địa chi mới, lần dò i ở phần tử cách khoảng i^2 cho đến khi gặp địa chỉ trống đầu tiên thì thêm phần tử vào địa chỉ này.

Bảng băm với phương pháp băm kép: bảng băm loại này dùng hai hàm băm khác nhau, băm lần đầu với hàm băm thứ nhất nếu bị xung đột thì xét địa chỉ khác bằng hàm băm thứ hai.

Ưu điểm của các Bảng băm:

Bảng băm là một cấu trúc dung hòa giữa thời gian truy xuất và dung lượng bộ nhớ:

- Nếu không có sự giới hạn về bộ nhớ thì chúng ta có thể xây dựng bảng băm với mỗi kháa ứng với một địa chỉ với mong muốn thời gian truy xuất tức thời.

- Nếu dung lượng bộ nhớ có giới hạn thì tổ chức một số khóa có cùng địa chỉ, lúc này thời gian truy xuất có bi suy giảm đôi chút.

Bảng băm được ứng dụng nhiều trong thực tế, rất thích hợp khi tổ chức dữ liệu có kích thước lớn và được lưu trữ ở bộ nhớ ngoài.

3. VÍ DỤ VỀ CÁC HÀM BĂM

Hàm băm dạng bảng tra:

Hàm băm có thể tổ chức ở dạng bảng tra (còn gọi là bảng truy xuất), 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, khóa b ứng với địa chỉ 1,… , z ứng với địa chỉ 25.

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	/	/

Hình 3.1 Hàm băm dạng bảng tra được tổ chức dưới dạng danh sách kề.

Hàm băm dạng công thức:

Thông thường hàm băm dạng công thức được xây dựng theo dạng tổng quát f(key).

Người ta thường dùng hàm băm chia dư (% modulo) như các ví dụ 1 và 2 sau:

Ví dụ 1: f(key) = key % 10:

Theo ví dụ này, hàm băm f(key) sẽ băm các số nguyên thành 10 địa chỉ khác nhau (ánh xạ vào các địa chỉ từ 0, 1,…, 9). Các khóa có hàng đơn vị là 0 được băm vào địa chỉ 0, các khóa có hàng đơn vị là i (i=0 | 1 | … | 9) được băm vào địa chỉ thứ i.

Ví dụ 2: f(key)=key % M:

Hàm băm loại này cho phép băm các số nguyên thành M địa chỉ khác nhau (ánh xạ vào các địa chỉ từ 0, 1,… M-1).

Ví dụ 3:

Giả sử cần xây dựng một hàm băm với tập khóa số là chuỗi 10 kí tự, tập địa chỉ có M địa chỉ khác nhau .

Có nhiều cách để xây dựng hàm băm này, ví dụ cộng dồn mã ASCII của từng kí tự, sau đó chia dư (% modulo) cho M.

Thông thường, hàm băm dạng công thức rất đa dạng và không bị ràng buộc bởi một tiêu chuẩn nào cả.

Yêu cầu đối với hàm băm tốt:

Một hàm băm tốt thường phải thỏa các yêu cầu sau:

- Phải giảm thiểu sự xung đột.
- Phải phân bố đều các phần tử trên M địa chỉ khác nhau của bảng băm.

2.4. CÁC CÁCH GIẢI QUYẾT XUNG ĐỘT

Như đã đề cập ở phần trên, sự xung đột là hiện tượng các khóa khác nhau nhưng băm cùng địa chỉ như nhau, hay ánh xạ vào cùng một địa chỉ

Một cách tổng quát, khi key1<>key2 mà f(key1)=f(key2) chúng ta nói phần tử có khóa key1 xung đột với phần tử có khóa key2.

Thực tế người ta giải quyết sự xung đột theo hai phương pháp: *phương pháp nối kết* và *phương pháp băm lại*.

Giải quyết sự xung đột bằng phương pháp nối kết:

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.

Các loại bảng băm giải quyết sự xung đột bằng phương pháp nối kết như: bảng băm với phương pháp nối kết trực tiếp, bảng băm với phương pháp nối kết hợp nhất.

Giải quyết sự xung đột bằng phương pháp băm lại:

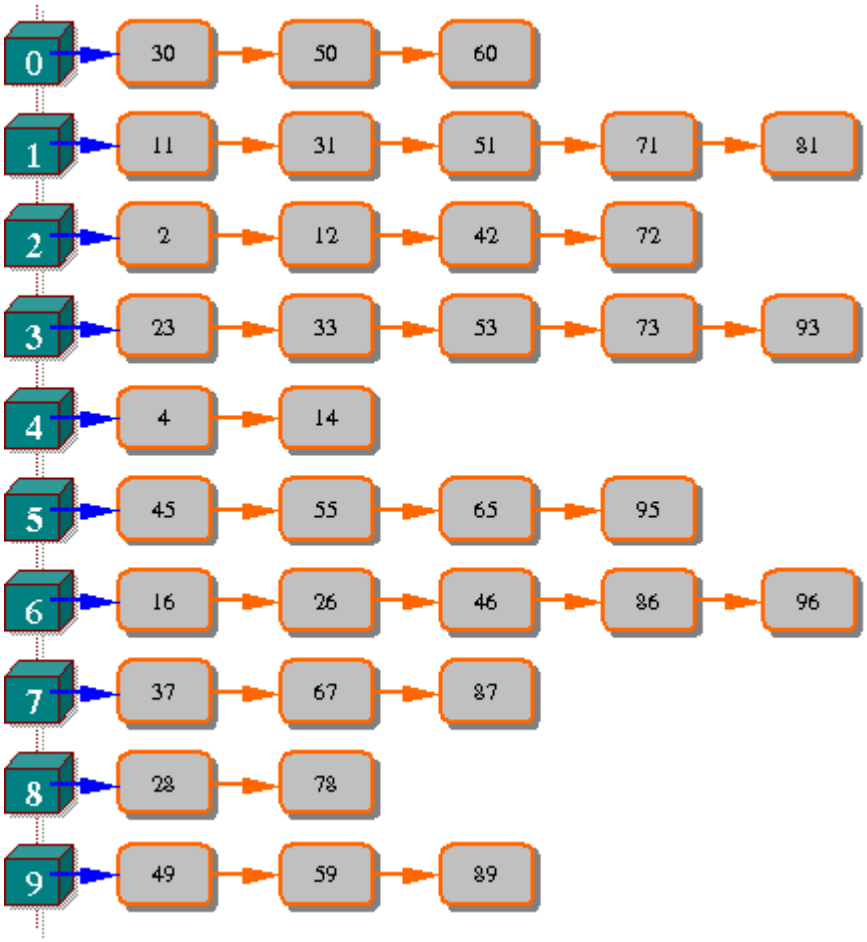
Nếu băm lần đầu bị xung đột thì băm lại lần 1, nếu bị xung đột nữa thì băm lai lần 2,… Quá trình băm lại diễn ra cho đến khi không còn xung đột nữa. Các pháp băm lại (rehash function) thường sẽ chọn địa chỉ khác cho các phần tử.

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.

Các loại bảng băm giải quyết sự xung đột bằng phương pháp băm lại như: bảng băm với phương pháp dò tuyến tính, bảng băm với phương pháp dò bậc hai, bảng băm với phương pháp băm kép.

2.4.1. Bảng băm với phương pháp nối kết trực tiếp (Direct chaining Method)

Mô tả: Xem hình vẽ



Hình 1.6. bảng băm với phương pháp nối kết trực tiếp

Bảng băm được cài đặt bằng các danh sách liên kết, các phần tử trên bảng băm được "băm" thành M danh sách liên kết (từ danh sách 0 đến danh sách M-1). Các phần tử bị xung đột tại địa chỉ i được nối kết trực tiếp với nhau qua danh sách liên kết i. Chẳng hạn, với M=10, các phần tử có hàng đơn vị là 9 sẽ được băm vào danh sách liên kết i = 9.

Khi thêm một phần tử có khóa k vào bảng băm, hàm băm f(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 f(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.

Để minh họa cho vấn đề vừa nêu:

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 f(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.

Cài đặt bảng băm dùng phương pháp nối kết trực tiếp :

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
Mbucket
*/
NODEPTR bucket[M];
```

b.Các phép toán:

Hàm băm

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

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

Chúng ta có thể dùng một hàm băm bất kì thay cho hàm băm dạng % trên.

Phép toán initbuckets:

Khởi động các bucket.

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

Phép toán emmptybucket:

Kiểm tra bucket b có bị rỗng không?

```
int emptybucket (int b)
{
```

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

- **Phép toán emmpty:**

Kiểm tra bảng băm có rỗng không?

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

- **Phép toán insert:**

Thêm phần tử có khóa k vào bảng băm.

Giả sử các phần tử trên các bucket là có thứ tự để thêm một phần tử khóa k vào bảng băm trước tiên chúng ta xác định bucket phù hợp, sau đó dùng phép toán place của danh sách liên kết để đặt phần tử vào vi trí phù hợp trên bucket.

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

- **Phép toán remove:**

Xóa phần tử có khóa k trong bảng băm.

Giả sử các phần tử trên các bucket là có thứ tự, để xóa một phần tử khóa k trong bảng băm cần thực hiện:

- Xác định bucket phù hợp
- Tìm phần tử để xóa trong bucket đã được xác định, nếu tìm thấy phần tử cần xóa thì loại bỏ phần tử theo các phép toán tương tự loại bỏ một phần tử trong danh sách liên kết.

```
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.

```
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.

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

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;
        }
}
```

Phép toán traverse:

Duyệt toàn bộ bảng băm.

```
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 hàm NULL,nếu tìm thấy hàm này trả về con trả 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
        // else //tim thay
        return(p);
}
```

Nhận xét bảng băm dùng phương pháp nối kết trực tiếp :

Bảng băm dùng phương pháp nối kết trực tiếp sẽ "băm" n phần tử vào danh sách liên kết (M bucket).

Để tốc độ thực hiện các phép toán trên bảng hiệu quả thì cần chọn hàm băm sao cho băm đều n phần tử của bảng băm cho M bucket, lúc này trung bình mỗi bucket sẽ có n/M phần tử. Chẳng hạn, phép toán *search* sẽ thực hiện việc tìm kiếm tuyến tính trên bucket nên thời gian tìm kiếm lúc này có bậc 0 (n/M) - nghĩa là, nhanh gấp n lần so với việc tìm kiếm trên một danh sách liên kết có n phần tử.

Nếu chọn M càng lớn thì tốc độ thực hiện các phép toán trên bảng băm càng nhanh, tuy nhiên lại càng dùng nhiều bộ nhớ. Do vậy, cần điều chỉnh M để dung hòa giữa tốc độ truy xuất và dung lượng bộ nhớ.

- Nếu chọn M=n thì năng xuất tương đương với truy xất trên mảng (có bậc O(1)), tuy nhiên tốn nhiều 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.

Chương trình minh họa:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#define TRUE 1
#definr FALSE 0
#define M 100
struct nodes
{
        int key;
        struct nodes *next;
};
typedef struct nodes *NODEPTR;
```

```
NODEPRT bucket[M];
//mang cac con tro chi nut dau cua cac bucket
//tac vu getnode(void)
{
        NODEPTR p;
        p = (NODEPTR) malloc(siseof(struct nodes));
        return(p);
}
//Tac vu freenode: huy nut da cap phat
void freenode(NODEPTR p)
{
        free(p);
}
// Ham bam
int hashfunc(int key)
{
        return(key % M);
}
//Khoi dong cac bucket
void initbucket( )
{
        int b;
        for (b = 0 ;b < M;b+)
        bucket[b] = NULL;
}
//Tac vu emptybucket;kiem tra but ket b co rong khong
int emptybucket (int b)
{
        return(bucket[b] == NULL? TRUE :FALSE);
}
//Tac vu empty:kiem tra bang bam co ranh khong
int empty( )
{
        int b;
        for (b=0;b<M;b++)
        if(bucket[b]!=NULL)
        return(FALSE);
        return(TRUE);
}
//Tac vu push;them nut moi vao au bucket b
void push(int b,int x)
{
        NODEPTR p;
        p = getnode( );
        p-> key = x;
        p-> next =bucket[b];
        bucket[b] = p;
}
//Tac vu pop: xoa nut o dau bucket b
int pop(int b)
{
        NODEPTR p;
        int k;
        int (emptybucket (b))
        {
```

```
                printf("\nBucket%d rong,khong xoa nut duoc ",b);
                return(0);
        }
        p = bucket[b]; //nut can xoa la nut dau but ket b
        k =p->key; //k la noi dung nut bi xoa
        bucket[b] = p->next;
        freenode(p);
        return(k);
}
//Tac vu insafter:them nut moi vao bucket sau nut p
void insafter(NODEPTR q, int k)
{
        NODEPTR q;
        if (p == NULL)
                printf("khong them nut moi duoc");
        else
        {
                q = getnode( );
                q->key = k;
                q-> key= p->next
                p->next=q;
        }
}
//Tac vu delafter:Xoa nut trong bucket trong nut p
int delafter(NODEPTR q)
{
        NODEPTR q;
        int k;
        if(p==NULL | | p->next == NULL)
        {
                printf ("khong xoa nut duoc"); return (0);
        }
        q=p->next; // q chi nut can xoa
        k =q ->key;//k la noi dung nut bi xoa
        p->next = q =next;
        freenode(q);
        return(k);
}
//Tac vu place:tac vu nay chi su dung khi them nut vao bucket da co thu tu
void place(int b,int k)
{
        NODEPTR p; q;//q la nut truoc ,p la nut sau
        q = NULL;
        for(p = bucket[b]; p!=NULL && k > p->key;
        p = p->next)
        q=p;
        if (q == NULL)//them nut vao dau buket
                push(b,k);
        else
                insafter (q, k);
}
//Tac vu insert;them nut co khoa k vao bang bam
void insert(int k)
{
        int b;
```

```
        b = hashfunc(k);
        place(b, k);//tac vu place cua danh sach lien ket*/
}
//Tac vu remove :xoa nut co khoa k trong bang bam
void remove (int k)
{
        int b;
        NODEPTR p, q;
        b=hashfunc(k);
        p=bucket[p];
        q=p;
        while(p !=NULL && p->key !=)
        {
                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 dann sach lien ket*/
        else
                delafter(q);
                /*tac vu delafter cua danh sach lien ket*/
}
//Tac vu clearbucket;xoa tat ca cac nut trong bucket b
void clearbucket (int b)
{
        NODEPTR p, q;// q la nut truoc , p la nut sau
        q=NULL;
        p=buket[b];
        while(p !=NULL)
        {
                q=p;
                p=p->next;
                freenode[b] = NULL;//khoi dong bucket b
        }
}
//Tac vu clear: xoa tat ca cac nut trong bang bam
void clear( )
{
        int b;
        for b=0; b<M; b++)
        clearbucket(b);
}
//Tac vu traversebucket:duyet bucket b
void traversebuket(int b)
{
        NODEPTR p;
        p=bucket[b];
        while (p !=NULL)
        {
                printf("%3d",p->key);
                p=p->next;
        }
```

```
}
//Tac vu traverse:duyet bang ham
void traverse( )
{
	int b;
	for (b=0;b<M; b++)
	{
		printf("\nBucket %d:",b);
		traversebucket(b);
	}
}
/*
Tac vu search: tim kiem mot nut trong bang bam, neu khong tim thay ham nay tra ve tri -1, neu tim thay ham
Tra ve 0
*/
int search(int k)
{
	NODEPTR p;
	int b;
	b = hashfunc(k);
	p = bucket[b];
	while(k >p->key && p !=NULL)
	p = =->next;
	if(p== NULL || k!=p->key) //khong tim thay
		return(-1);
	else // tim thay
	return 1;
}
/*
Chuong trinh chinh
*/
Void main( )
{
	intb,key,i,n,chucnang;
	char c;
	clrscr();
	initbucket(); //khoi dong M bucket cua bang bam
	do
	{
		//Menu chinh cua chuong trinh
		printf("\n\Cac chuc nang cua chuong trinh:\n");
		printf("\1:Them mot nut vao bang bam\n");
		printf("\2:Them ngau nhien nhieu nut vao bang bam\n");
		printf("\3: Xoa nut trong bang bam\n");
		printf("\4: Xoa toan bo bang bam\n");
		printf("\5: Duyet bang bam\n");
		printf("\6: Tìm kiem tren bang bam\n");
		printf("\0:Ket thuc chuong trinh\n");
		printf("\n Chuc nang ban chon:");
		scanf("&d",& chuc nang);
		switch(chuc nang) {
			case 1:
			{
				printf("\nTHEM MOT NUT VAO BANG BAM");
				printf("\ Khoa cua nut moi:");
```

```
                        scanf("%d;,&key);
                        insert(key);
                        break;
                }
                case 2:
                {
                        printf("\nTHEM NGAU HIEN NHIEU NUT VAO BANG BAM");
                        printf("\n Ban muon them bao nhieu nut:");
                        scanf("%d",&n);
                        for (i=0;i<n;i++) {
                                key = random(100);
                                insert(key);
                        }
                        break;
                }
                case 3:
                {
                        printf("\nXoa TREN BANG BAM");
                        printf("\n khoa cua nut can xoa:");
                        scanf("%d",&key);
                        remove(key);
                        break;
                }
                case 4:
                {
                        printf("\nXoa TOAN BO BANG BAM");
                        printf("\nban co chac chan khong (c/k):");
                        c=getch();
                        if(c== 'c' | | c == 'c')
                        clear( );
                        break;
                }
                case 5:
                {
                        printf("\n DUYET BANG BAM");
                        traverse( );
                        break;
                }
                case 6:
                {
                        printf("\nTIM KIEM TREN BANG BAM");
                        pintf("\n Khao can tim:");
                        scanf("%d",&key);
                        b=search(key);
                        if(b == -1)
                                printf(" khong thay");
                        else
                                printf(" Tim thay trong bucket d",b);
                        break;
                }
        }
        while(chucnang !=0);
        clear( ); //Xoa tat ca cac nut tren bang bam
}
```

2.4.2. Bảng băm với phương pháp nối kết hợp nhất (Coalesced chaining Method)

Mô tả:

- *Cấu trúc dữ liệu:* Tương tự như trong trường hợp cài đặt bằng phương pháp nối kết trực tiếp, bảng băm trong trường hợp này được cài đặt bằng danh sách liên kết dùng mảng, có M phần tử. Các phần tử bị xung đột tại một địa chỉ được nối kết nhau qua một danh sách liên kết. Mỗi phần tử của bảng băm gồm hai trường:

  - Trường key: chứa khóa của mỗi phần tử
  - Trường next: con trỏ chỉ đến phần tử kế tiếp nếu có xung đột.

- *Khởi động*: Khi khởi động, tất cả trường key của các phần tử trong bảng băm được gán bởi giá trị Null, còn tất cả các trường next được gán -1.

- *Thêm mới một phần tử*: Khi thêm mới một phần tử có khóa key vào bảng băm, hàm băm f(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến M-1.

  - Nếu chưa bị xung đột thì thêm phần tử mới vào địa chỉ này.
  - Nếu bị xung đột thì phần tử mới được cấp phát là phần tử trống phía cuối mảng. Cập nhật liên kết next sao cho các phần tử bị xung đột hình thành một danh sách liên kết.

- *Tìm kiếm*: Khi tìm kiếm một phần tử có khóa key trong bảng băm, hàm băm f(key) sẽ giúp giới hạn phạm vi tìm kiếm bằng cách xác định địa chỉ i trong khoảng từ 0 đến M-1, và việc tìm kiếm phần tử khóa có khoá key trong danh sách liên kết sẽ xuất phát từ địa chỉ i.

Để minh họa cho bảng băm với phương pháp nối kết hợp nhất, xét ví dụ sau:

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 f(key) = key % 10.

Key : A C B D E

Hash: 1 2 1 1 3

key	**next**
NULL	-1
NULL	-1
...	...
NULL	-1

0	NULL	-1
1	A	M-1
2	C	-1
3	E	-1
...	...	
M-2	D	-1
M-1	B	M-2

Cài đặt bảng băm dùng phương pháp nối kết hợp nhất:

a. 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 nut cua bang bam
struct node
{
	int key; //khoa cua nut tren bang bam
	int next;
	//con tro chi nut ke tiep khi co xung dot
};
//Khai bao bang bam
struct node hashtable[M];
int avail;
/*
bien toan cuc chi nut trong o cuoi table duoc cap nhat khi co xung dot
*/
```

b. Các tác vụ:

Hàm băm:

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

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

Chúng ta có thể dùng một hàm băm bất kì thay cho hàm băm dạng % trên.

Phép toán khởi tạo (Initialize):

Phép toán này cho khởi động bảng băm: gán tất cả các phần tử trên bảng có trường key là Null, trường next là -1.

Gán biến toàn cục avail=M-1, là phần tử cuối danh sách chuẩn bị cấp phát nếu xảy ra xung đột.

```
void initialize()
{
	int i;
	for(i = 0;i<M;i++)
	{
		hashtable[i].key = NULLKEY;
		hashtable[i].key = -1;
	}
	avail =M-1;
	/* nut M-1 la nut o cuoi bang chuan bi cap phat neu co xung dot*/
}
```

Phép toán kiểm tra rỗng (empty):

Kiểm tra bảng băm có rỗng không.

```
int empty ();
{
	int i;
	for(i = 0;i< M;i++)
	if(hashtable[i].key !=NULLKEY)
		return(FALSE);
	return(TRUE);
```

```
}
```

- **Phép toán tìm kiếm (search):**

Tìm kiếm theo phương pháp tuyến tính, nếu không tìm thấy hàm tìm kiếm trả về trị M, nếu tìm thấy hàm này trả về địa chỉ tìm thấy.

```
int search(int k)
{
	int i;
	i=hashfunc(k);
	while(k !=hashtable[i].key && i !=-1)
		i=hashtable[i].next;
	if(k== hashtable[i]key)
		return(i);//tim thay
	return(M);//khong tim thay
}
```

- **Phép toán lấy phần tử trống (Getempty):**

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

```
int getempty()
{
	while(hashtable[avail].key !=NULLKEY)  avail - -;
	return(avail);
}
```

- **Phép toán chèn phần tử mới vào bảng băm (insert):**

Thêm phần tử có khóa k vào bảng băm.

```
int insert(int k)
{
	int i;
	//con tro lan theo danh sach lien ket chua cac nut //bi xung dot
	int j;
	//dia chi nut trong duoc cap phat
	i = search(k);
	if(i !=M)
	{
		printf("\n khoa %d bi trung,khong them nut nay duoc",k);
		return(i);
	}
	i=hashfunc(k);
	while(hashtable[i]next >=0) i=hashtable[i].next;
	if(hashtable[i].key == NULLKEY)
		//Nut i con trong thi cap nhat
		j = i;
	else
	//Neu nut i la nut cuoi cua DSLK
	{
		j = getempty();
		if(j < 0)
		{
			printf("\n Bang bam bi day,khongthem nut co khoa %d duoc"k);
			return(j);
		}
		else
			hashtable[i].next = j;
	}
	hashtable[j].key = k;
	return(j);
}
```

Nhận xét bảng băm dùng phương pháp nối kết hợp nhất:

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 0(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 0(n).

Chương trình minh họa:

Chương trình Hashtable, dùng phương pháp nối kết hợp nhất *(coalesced chaining method*) - Cài đặt bằng danh sách kề.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define TRUE 1
#define FALSE 0
#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 nut cua bang bam
struct node
{
	int key; //khoa cua nut tren bang bam
	int next; //con tro chi nut ke tiep khi co xung dot
};
//Khai bao bang bam
struct node hashtable[M];
int avail;
//bien toan cuc chi nut trong o cuoi table duoc cap phat khi co xung dot
//Ham bam
int hashtable(int key)
{
	return(key % M);
}
//Khoi dong bang bam
void initialize()
{
	int i;
	for (i=0; i<M;i++)
	{
		hashtable[i].key=NULLKEY;
		hashtable[i].next=-1;
	}
	avail=M-1;
	//nut M-1 la nut o cuoi bang chuan bi cap phat nut co xung dot
}
//Tac vu empty: kiem tra bang baam co ranh khong
int empty()
{
	int i;
	for (i= 0;i<M;i++)
	if(hashtable[i].key !=NULLKEY)
		return(FALSE);
	return(TRUE);
}
```

```
/*Tac vu search: tim kiem theom phuong phap tuyen tinh , neu khong tim thay ham nay tra ve vi tri M,
neu tim thay ham nay tra ve dia chi tim thay
*/
int search(int k)
{
        int i;
        i= hashfunc(k);
        while(k !=hashtable[i].key && 1 !=-1)
                i = hashtable[i].next;
        if(k == hashtable[i].key; //Tim thay
                return(i);
        else
        //khong tim thay
        return(M);
}
/*Ham getempty: chon nut con trong phia cuoi hashtable de cap nhat khi xay ra xung dot
*/
int getempty()
{
        while(hashtable[avail].key !=NULLKEY)
                avail--;
        return(avail);
}
//Tac vu insert: them nut co khoa k vao bang bam
int insert (int k)
{
        int i;
        //con tro lan theo danh sach lien ket chua cac nut //bi xung dot
        int j;
        //dia chi nut trong duoc cap phat
        i = search(k);
        if(i !=M)
        {
                printf("\n khoa %d bi trung, khong them nut nay duoc", k);
                return(i);
        }
        i = hashfunc(k0;
        while(hashtablr[i].next >=0)
        i = hashtable[i].next;
        if(hashtable[i].key ==NULLKEY)
                //Neu nut i con trong thi cap phat
                j=i;
        else
        {
                //Neu nut i la nut cuoi cua DSLK
                j=getempty();
                if(j < 0)
                {
                        printf("\n Bang bam bi day khong them nut co khoa % d duoc:", k);
                        return(j);
                }
                else
                        hashtable[i],next=j;
        }
        hashtable[j].key=k;
        return(j);
```

```
}
//Tac vu viewtable:xem chi tiet bang bam
void viewtable()
{
        int i;
        for(i= 0;i < M; i++)
        printf("\ntable[%2d]: %4d",i,hashtable[i].next);
}
//Chuong trinh chinh
void main( )
{
        int i,n,p,q;
        int b,key,chucnang;
        char c;
        clrscr();
        //Khoi dong bang bam
        initialize();
        do
        {
                //Menu chinh cua chuong trinh
                printf("\n\nCac chuc nang cua chuong trinh:\n");
                printf("1: Them nut moi vao bang bam\n");
                printf("2: Them ngau nhien nut vao bang bam\n");
                printf("3: Xoa toan bo bang bam\n");
                printf("4: Xem chi tiet bang bam\n");
                printf("5 : Tim kiem tren bang bam\n");
                printf("0: Ket thuc chuong trinh\n");
                printf("\nChuc nang ban chon:");
                scanf("%d", & chucnang);
                switch(chucnang)
                {
                        case 1:{
                                printf("\nTHEM NUT MOI VAO BANG BAM");
                                printf("\n Khoa cua nut moi:");
                                scanf("%d",&key);
                                insert(key);
                                break;
                        }
                        case 2:{
                                printf("\n Them ngau nhien nut vao bang bam");
                                printf(\n Ban muon them bao nhieu nut:");
                                scanf("%d",&key);
                                for(i=0;i<n;i++)
                                {
                                        key=random(1000);
                                        insert(key);
                                }
                                beark;
                        }
                        case 3: {
                                printf("\n XOA TOAN BO BANG BAM");
                                printf("\N BAN CO CHAC CHAN KHONG (C/K):");
                                c=getch();
                                if(c=='c' | | c =='c')
                                initialize( );
                                beark;
```

```
                        }
                        case 4:{
                                printf("\n XEM CHI TIET BANMG BAM:");
                                viewtable();
                                break;
                        }
                        case 5:{
                                printf("\nTIM KIEM TREN BANG BAM:");
                                printf("\n Khoa can tim:");
                                scanf("%d",&key);
                                if(search(key0=M)
                                        printf("khongtim thay");
                                else
                                        printf("Tim thay tai dia chi %d trong bang bam",
                                search(ke y )) ;
                                beark;
                        }
                }
        }while(chucnang !=0);
}
```

2.4.3. Bảng băm với phương pháp dò tuyến tính (Linear Probing Method)

Mô tả:

- *Cấu trúc dữ liệu:* 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 chứa khoá của phần tử.

Khi khởi động bảng băm thì tất cả trường key được gán Null

- *Khi thêm phần tử* có khoá key vào bảng băm, hàm băm f(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến M-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 f_1 sẽ xét địa chỉ kế tiếp, nếu lại bị xung đột thì hàm băm thì hàm băm lại lần 2, hàm f_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.

- *Khi tìm một phần tử* có khoá key trong bảng băm, hàm băm f(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến M-1, tìm phần tử khoá key trong khối đặt chứa các phần tử xuất phát từ địa chỉ i.

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:

f(key)=(f(key)+i) %M với f(key) là hàm băm chính của bảng băm.

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

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 f(key) = key % 10.

Hình thể hiện thêm các nut 32, 53, 22, 92, 17, 34, 24, 37, 56 vào bảng băm.

0	NULL	0	NULL	0	NULL	0	NULL	0	56
1	NULL	1	NULL	1	NULL	1	NULL	1	NULL
2	32	2	32	2	32	2	32	2	32
3	53	3	53	3	53	3	53	3	53
4	NULL	4	22	4	22	4	22	4	22
5	NULL	5	92	5	92	5	92	5	92
6	NULL	6	NULL	6	34	6	34	6	34
7	NULL	7	NULL	7	17	7	17	7	17
8	NULL	8	NULL	8	NULL	8	24	8	24
9	NULL	9	NULL	9	NULL	9	37	9	37

Cài đặt bảng băm dùng phương pháp dò tuyến tính:

a. 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*/
```

b. Các tác vụ:

- **Hàm băm:**

Giả sử chúng ta chọn hàm băm dạng%:f(key0=key %10.

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

Chúng ta có thể dùng một hàm băm bất kì thay cho hàm băm dạng % trên.

- **Phép toán khởi tạo (initialize):**

Khởi tạo bảng băm.

Gán tất cả các phần tử trên bảng có trường key là NULL.

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:**

  Việc tìm kiếm phần tử có khoá k trên một khối đặc, bắt đầu từ một địa chỉ i = HF(k), nếu không tìm thấy phần tử có khoá k, hàm này sẽ trả về trị M, còn nếu tìm thấy, hàm này trả về địa chỉ tìm thấy.

```
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)+) % 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);
}
```

- **Nhận xét bảng băm dùng phương pháp dò tuyến tính:**

  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 kẻ nhau, tốc độ truy xuất lúc này có bậc 0(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 0(n).

Chương trình minh họa:

Bảng băm, dùng phương pháp dò tuyến tính (linear proping method)-cài đặt bằng danh sách kề.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define TRUE 0
#define FALSE -1
#define NULLKEY ◆1
#define M 100
//Khai bao cau truc mot nut cua bang bam
struct node
{
        int key;//khoa cua nut tren 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
//Ham bam
int hashfunc(int key)
{
        return(key % M);
}
//Khoi dong bang bam
void initialize( )
{
        int i;
        for (i=0;i <M;i++)
        hashtable[i].key=NULLKEY;
        N= 0;//so nut hien co khoi dong bang 0
}
//Tac vu empty:kiem tra bả bang bam co ranh khong
int empty( )
{
        return(N ==0 ?TRUE :FALSE);
}
//Tac vu full:kiem tra bang bam da day chua
int full( )
{
        return (N == M-1 ? TRUE :FALSE);
```

```
}
/* Tac vu search: tim kiem nut co khoa k tren bang bam, neu khong tim thay ham nay tra vê vi tri M,
neu tim thay ham nay tra ve đia chi tim thay
*/
int search (int k)
{
        int i;
        i= hashfunc(k);
        while(hashtable[i].key!=k&&hashtable[i].key !=NULLKEY)
        {
                //bam lai(theo phuong phap do tim tuyen tinh):hi(key)=h(key)+i)% M)
                i=i+1;
                if(i >=M)
                i=i-M;
        }
        if(hashtable[i].key ==k)//tim thay
                rerurn(i);
        else//khong tim thay
        return(M);
}
//Tac vu insert:them nut co khoa k vao bang bam
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);
}
//Tac vu remove:xoa nut tai dia chi i tren bang bam
Void remove(int i)
{
        int j, r, cont, a;
        cont = TRUE;
        do
        {
                hashtable[i].key = NULLKEY;
                j = i;
                do
                {
                        i=i +1;
                        if(i >=M)
                        i=i -M;
                        if(hashtable[i].key == NULLKEY)
                                cont = FALSE;
```

```
                        else
                        {
                                r = hashfunc(hashtable[i].key);
                                a = (j <r && r <=i) | | (r <=i && i <j) | | (i<j && j <r);
                        }
                }
                while (cont && a);
                if(cont)    hashtable[j].key=hashtable[i].key;
        }while(cont);
}
//Tac vu viewtable:xem chi tiet bang bam
Void viewtable()
{
        int i;
        for(i=0; i <M;i++)
                printf("\ntable[%2s]: %4d",i,hashtable[i].key);
}
// Chuong trinh chinh
main( )
{
        int i,n,p,q;
        int b,key,chucnang;
        char C;
        clrscr( );
        //Khoi tao bang bam
        initiallize( );
        do
        {
                //Menu chinh cua chuong trinh
                printf("\n\nCac chuc nang cua chuong trinh:\n");
                printf("1: Them nut moi vao bang bam\n");
                printf("2: Them ngau nhien nhieu nut vao bang bam\n");
                printf("3: Xoa nut tren bang bam\n");
                printf("4: Xoa toan bo bang bam\n");
                printf("5: Xem chi tiet bang bam\n");
                printf("6:Tim kiem tren bang bam\n");
                printf("0: Ket thuc chuong trinh\n");
                printf("\nChuc nang ban chon:");
                scanf("%d", &chucnang);
                switch(chucnang)
                {
                        case 1:
                        {
                                printf("\nTHEM NUT VAO BANG BAM");
                                printf("\n Khoa cua nut moi:");
                                scanf("%d",&key);
                                break;
                        }
                        case 2:
                        {
                                printf("\nTHEM NGAU NHIEN NHIEU NUT VAO BANG BAM");
                                printf("\n Ban muon them bao nhieu nut:");
                                scanf("%d",&n);
                                for(i=0i <n;i ++)
                                {
                                        key = random(1000);
```

```
                        insert(key);
                }
                break;
        }
        case 3:
        {
                printf("\nXOA NUT TREN BANG BAM");
                printf("\n Khoa cua nut can xoa:");
                scanf("%d",&key);
                i =search(key);
                if(i ==M);
                        printf("Khong co nut voi khoa can xoa");
                else
                {
                        remove(i);
                        N--;
                }
                break;
        }
        case 4:
        {
                printf("\n XOA TOAN BO BANG BAM");
                printf("\n Ban co chac khong (c/k):");
                c = getch( );
                if(c = =”c” | | == “c”)
                initialize( );
                break;
        }
        case 5:
        {
                printf("\nXEM CHI TIET BANG BAM");
                viewtable( );
                break;
        }
        case 6:
        {
                printf("\n TIM KIEM TREN BANG BAM");
                printf("\n Khoa can tim:");
                if(search(key) == M)
                        printf(" khong thay");
                else
                        printf(" tim thay tai dia chi %d trong bang bam",search(key));
                beark;
        }
    }
    scanf("%d",&key);
}while(chucnang !=0);
return(0);
}
```

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

Mô tả:

- *Cấu trúc dữ liệu:* Bảng băm dùng phương pháp dò tuyến tính bị hạn chế do rải các phần tử không đều, bảng băm với phương pháp dò bậc hai rải các phần tử đều hơn.

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 chứa khóa các phần tử.

- *Khi khởi động* bảng băm thì tất cả trường key bị gán NULL.

Khi thêm phần tử có khóa key vào bảng băm, hàm băm f(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến M-1.

- Nếu chưa bị xung đột thì thêm phần tử mới vào địa chỉ i này.
- Nếu bị xung đột thì hàm băm lại lần 1 f1 sẽ xác định địa chỉ cách 1^2, nếu lại bị xung đột thì hàm băm lại lần 2 f2 sẽ xét địa chỉ cách i 2^2 ,… , quá trình cứ thế cho đến khi nào tìm được trống và thêm phần tử vào địa chỉ này.

- *Khi tìm kiếm* một phần tử có khóa key trong bảng băm thì xét phần tử tại địa chỉ i=f(key), nếu chưa tìm thấy thì xét phần tử 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).

- *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:

fi(key)=(f(key) + i^2) % M

với f(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.

Bảng băm minh họa 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 f(key) = key % 10.

Cài đặt bảng băm dùng phương pháp dò bậc hai:

a. 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
```

b. Các tác vụ :

Hàm băm:

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

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

Chúng ta có thể dùng một hàm băm bất kì tahy cho hàm băm dạng % trên.

- **Phép toán initialize**

Khởi động hàm băm.

Gán tất cả các phần tử trên bảng có trường key là NULLKEY.

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 search(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);
}
```

Nhận xét bảng băm dùng phương pháp dò bậc hai:

- Nên chọn số địa chỉ M là số nguyên tố. Khi khởi động bảng băm thì tất cả M trường key được gán NULL, 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 0(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.

2.4.5. Bảng băm với phương pháp băm kép (Double hashing Method)

Mô tả:

- *Cấu trúc dữ liệu:* 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 kì, ví dụ chọn hai hàm băm như sau:

f1(key)= key %M.

f2(key) =(M-2)-key %(M-2).

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.

- *Khi khởi động* bảng băm,tất cả trường kay được gán NULL.

- *Khi thêm phần tử* có khoá key vào bảng băm, thì i=f1(key) và j=f2(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 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.

- *Khi tìm kiếm* một phần tử có khoá key trong bảng băm, hàm băm i=f1(key) và j=f2(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+ji+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).

Bảng băm dùng hai hàm băm khác nhau, hàm băm lại của phương pháp băm kép được tính theo I (từ hàm băm thứ nhất) và j (từ hàm băm thứ hai) theo một công thức bất kì, ở đây minh họa bằng địa chỉ mới cách j. 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.

Bảng băm minh họa 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 f(key) = key % 10.

Minh hoạ:

Sau đây là minh hoạ cho bảng băm có tập khóa là tạâp số tự nhiên,tập địa chỉ có 11 địa chỉ (M=11)(từ địa chỉ 0 đến 10),chọn hàm băm f1(key)=key % 10 và f2(key)=9-key %9.

Xem việc minh hoạ này như một bài tập.

Cài đặt bảng băm dùng phương pháp băm kép:

a. 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 phan tu 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
```

b. Các tác vụ:

- **Hàm băm:**

Giả sử chúng ta chọn hai hàm băm dạng %:

f1(key0=key %M va f2(key) =M-2-key%(M-2).

```
//Ham bam thu nhat
int hashfunc(int key)
{
        return(key%M);
}
//Ham bam thu hai
int hashfunc2(int key)
{
        return(M-2 - key%(M-2));
}
```

Chúng ta có thể dùng hai hàm băm bất kỳ thay cho hai hàm băm dạng % trên.

- **Phép toán initialize :**

Khởi động bảng băm.

Gán tất cả các phần tử trên bảng có trường key là NULL.

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 dành ít nhất một phần tử trống trên bảng băm.

- **Phép toán search :**

Tìm kiếm phần tử có khóa k trên bảng băm, nếu không tìm thấy hàm này trả về về trị M, nếu tìm thấy hàm này trả về địa chỉ tìm thấy.

```
int search(int k)
{
        int i, j ;
        i = hashfunc (k);
        j = hashfunc2 (k);
        While (hashtable [i].key!=k && hashtable [i] .key ! = NULLKEY)
        //bam lai (theo phuong phap bam kep)
                i = (i+j) % 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 ("Bang bam bi day") ;
                return (M) ;
        }
        if (search (k) < M)
        {
                printf ("Da co khoa nay trong bang bam") ;
                return (M) ;
        }
        i = hashfunc (k) ;
        j = hashfunc 2 (k) ;
        while (hashtable [i].key ! = NULLEY)
                // Bam lai (theo phuong phap bam kep)
                i = (i + j) % M;
        hashtable [i].key = k ;
        N = N+1;
        return (i) ;
}
```

Nhận xét bảng băm dùng phương pháp băm kép:

- 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.

5. TỔNG KẾT VỀ PHÉP BĂM

- Bảng băm đặt cơ sở trên mảng.

- Phạm vi các giá trị khóa thường lớn hơn kích thước của mảng.
- Một giá trị khóa được băm thành một chỉ mục của mảng bằng hàm băm.
- Việc băm một khóa vào vào một ô đã có dữ liệu trong mảng gọi là sự đụng độ.
- Sự đụng độ có thể được giải quyết bằng hai phương pháp chính: Phương pháp nối kết và phương pháp băm lại.
- Trong phương pháp băm lại, các mục dữ liệu được băm vào các ô đã có dữ liệu sẽ được đưa vào ô khác trong mảng.
- Trong phương pháp nối kết, mỗi phần tử trong mảng có một danh sách liên kết. Các mục dữ liệu được băm vào các ô sẽ được đưa vào danh sách ở ô đó.

Vấn đề Hàm băm

- Hàm băm dùng phương pháp chia: h(k) = k mod m
- m là kích thước bảng băm, k là khóa.
- Hàm băm dùng phương pháp nhân: h(k) = $\lfloor m(k A \bmod 1) \rfloor$
- Knuth đề nghị A = 0.6180339887

Số lần đụng độ: (ví dụ)

Kích thước bảng băm	PP chia	PP Nhân
200	698	699
512	470	466
997	309	288
1024	301	292

- Theo bảng trên kết quả cho thấy kích thước bảng băm tỷ lệ nghịch với số lần đụng độ. Số đụng độ còn phụ thuộc vào phương pháp sử dụng hàm băm.
- ***Hệ số tải*** *là tỉ số giữa các mục dữ liệu trong một bảng băm với kích thước của mảng.*
- *Hệ số tải cực đại trong phương pháp băm lại khoảng 0,5. Đối với băm kép ở Hệ số tải này (0,5), các phép tìm kiếm sẽ có chiều dài thăm dò trung bình là 2.*
- Trong phương pháp băm lại , thời gian tìm kiếm sẽ là vô cực khi hệ số tải đạt đến 1.
- Điều quan trọng trong phương pháp băm lại là bảng băm không bao giờ được quá đầy.
- Phương pháp nối kết thích hợp với hệ số tải là 1.
- Với hệ số tải này, chiều dài thăm dò trung bình là 1,5 khi phép tìm thành công, và là 2.0 khi phép tìm thất bại.
- Chiều dài thăm dò trong phương pháp nối kết tăng tuyến tính theo hệ số tải.

- Kích thước của bảng băm thường là số nguyên tố. Điều này đặc biệt quan trọng trong thăm dò bậc hai và trong phương pháp nối kết.
- Các bảng băm có thể dùng cách lưu trữ ngoại. Một cách để thực hiện việc này là cho các phần tử trong bảng băm chứa số lượng các khối của tập tin trên đĩa

Chương trình từ điển cài đặt theo phương pháp kết nối trực tiếp

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
#include<conio.h>
#include<stdlib.h>
#include<alloc.h>
#define TRUE 1
#define FALSE 0
#define M 26
typedef struct node
{
        char word [10] ;
        char mean[50];
        struct node *Next;
}NodeType;
typedef NodeType *NodePtr;
NodePtr bucket[M];

NodePtr GetNode(char word[], char mean[])
{
        NodePtr p;
        p=(NodePtr) malloc(sizeof (NodeType));
        strcpy(p->word,word);
        strcpy(p->mean,mean);
        p->Next=NULL;
        return (p);
}
/*giải phóng một nút p ra khỏi tự điển*/
void freenode (NodePtr p)
{
        free (p);
}
/********************/
int hashfunc(char word [] )
{
        char ch=toupper(word[0]);
        return ((ch-65) % M);
}
/*khởi tạo thùng bucket*/
void initbucket ()
{
        int b;
        for (b=0;b<M;b++)
        bucket[b] =NULL;
}
/**thêm một nút I vào vào thùng bucket**/
void Insert (NodePtr p)
{
        int i=hashfunc(p->word);
```

```
        p->Next=bucket[i];
        bucket[i]=p;
}
/*********************/
NodePtr Find(char word[])
{
        int done =1;
        NodePtr temp;
        int i=hashfunc(word);
        temp=bucket[i];
        while (done && temp!=NULL)
        {
                if(strcmp(temp->word,word)==0) done=0;
                else temp=temp->Next;
        }
        if(done ==0)
                return temp;
        else return NULL;
}
/**hàm tạo từ điển**/
void MakeDictionary()
{
        NodePtr p;
        char word[10];
        char mean[50];
        do
        {
                fflush(stdin);
                printf("\n Nhập từ cần tra :");
                gets(word);
                if(!strcmp(word,"")) break;
                fflush(stdin);
                printf("\n%d Nhập nghiã :",hashfunc(word));
                gets(mean);
                p=GetNode(word,mean);
                Insert(p);
        }
        while (1);
}

/***hàm tìm một từ trong từ điển****/
void FindWord()
{
        NodePtr p;
        char word[10];
        printf("\n Nhập từ: ");
        fflush(stdin);
        gets(word);
        p=Find(word);
        if(p==NULL) printf("Không có trong từ điển");
        else printf("\n Có từ: %s \nNghiã là %s \n ", p->word,p->mean);
        return;
}
void PrintList(NodePtr List)
{
        NodePtr temp;
```

```
        temp=List;
        while (temp!=NULL)
        {
                printf("\n Từ: %s",temp->word);
                printf("\n Nghiã: %s\n\n",temp->mean);
                temp=temp->Next;
        }
}
void DisplayDictionnary()
{
        int i;
        for (i=0;i<M;i++)
        PrintList(bucket[i]);
}

/***** chương trình chính****/
void main()
{
        int chon;
        do
        {
                clrscr();
                printf(" \n\t\t CHƯƠNG TRÌNH TẠO MỘT TỪ ĐIỂN");
                printf(" \n1.XÂY DỰNG TỪ ĐIỂN");
                printf(" \n2. TRA TỪ");
                printf(" \n3. XEM TOÀN BỘ TỪ ĐIỂN");
                printf(" \n4. Quit");
                printf("\n bạn chọn chức năng nào:");
                scanf("%d",&chọn);
                switch (chọn)
                {
                        case 1: MakeDictionary(); break;
                        case 2: FindWord(); break;
                        case 3: DisplayDictionnary(); break;
                }
                getch();
        }
        while (chọn!=4);
        return 0;
}
```