Chương 4:
Bảng Băm (Hash table)

Mục tiêu

- Kiểu dữ liệu tự điển là tập các đối tượng dữ liệu mà trên đó ta quan tâm đến 3 phép toán cơ bản: tìm kiếm, chèn và hủy.
- Chương này tiếp cận một cấu trúc dữ liệu cài đặt hiệu quả kiểu dữ liệu từ điển, đó là: Bảng băm

NỘI DUNG

I. Giới thiệu chung về bảng băm
II. Các hàm băm
III. Các phương pháp xử lý xung đột
IV. Ứng dụng từ điển

I. Giới thiệu chung về bảng băm

1. Giới thiệu bảng băm
2. Cấu trúc bảng băm
3. Các phép toán trên bảng băm
4. Hàm băm
5. Vấn đề xung đột
6. Giải quyết xung đột
7. Phân loại bảng băm

1. Giới thiệu bảng băm (hash table):

- Các phép toán trên cấu trúc dữ liệu như danh sách, cây nhị phân … được thực hiện bằng cách so sánh phần tử của cấu trúc. Do đó, thời gian truy xuất không nhanh và phụ thuộc vào kích thước cấu trúc.
- Bảng băm giúp hạn chế số lần so sánh, giảm thiểu thời gian truy xuất. Độ phức tạp của các phép toán trên bảng băm thường bậc O(1) và không phụ thuộc vào kích thước của bảng băm.

- Bài toán: cần lưu trữ các mẫu tin và thực hiện các thao tác
  - Thêm mẫu tin
  - Xoá mẫu tin
  - Tìm mẫu tin theo khóa
- Tìm cách thức thực hiện một cách hiệu quả?

6

- Mảng chưa sắp xếp (Unsorted array)
  - Sử dụng mảng để lưu mẫu tin, không có thứ tự
  - Thêm: thêm cuối nhanh O(1)
  - Xoá: chậm do tìm rồi xoá O(n)
  - Tìm kiếm: tuần tự chậm O(n)
- Mảng đã sắp xếp (Sorted array)
  - Sử dụng mảng lưu trữ mẫu tin, có thứ tự
  - Thêm: chèn vào đúng vị trí, chậm O(n)
  - Xoá: phải dời các phần tử phía sau, chậm O(n)
  - Tìm: nhị phân, nhanh O(logn)

7

- Danh sách liên kết (Linked list)
  - Lưu trữ mẫu tin trong danh sách liên kết
  - Thêm đầu, cuối: nhanh, O(1)
  - Xoá đầu: nhanh O(1)
  - Tìm kiếm: tìm kiếm tuần tự O(n)
- Cấu trúc dữ liệu phức tạp hơn, nhưng thực thi tốt hơn
  - Tree BST
  - Hash table

8

Ví dụ:

0012345	An	8.15
0033333	Binh	90
0056789	Danh	5.68
...		
9801010	Phuong	2.0
9802020	Minh	10.0
...		
9903030	Thao	7.3
9908080	Tung	4.9

Vấn đề: lưu trữ 1,000 mẫu tin sinh viên và tìm kiếm theo mã số sinh viên.

9

Giải pháp 1

- Lưu trữ mẫu tin trong mảng lớn: chỉ mục tương đương khóa
- Thêm: rất nhanh O(1)
- Xóa: rất nhanh O(1)
- Tìm: rất nhanh O(1)
- Nhưng lãng phí rất nhiều bộ nhớ!

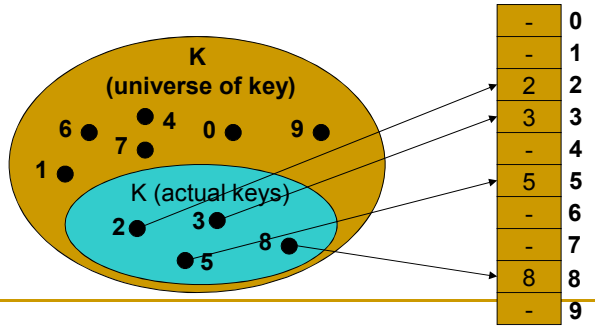
Có thể lưu trữ mẫu tin trong mảng cực lớn 0..9999999
Index được sử dụng như là mã số sinh viên. Khi đó mẫu tin sv với ms 0012345 được lưu trữ ở A[12345]!

0		
:	:	:
12345	An	8.15
:	:	:
33333	Binh	9.0
:	:	:
56789	Danh	5.68
:	:	:
:	:	:
9908080	Tung	4.9
:	:	:
9999999		

10

Giải pháp 2

- Dạng bảng băm với địa chỉ trực tiếp
  - Mỗi vị trí tương ứng một khoá trong K
  - Nếu 1 phần tử x có khoá k, thì T[k] chứa con trỏ đến x
  - Ngược lại T[k] = Ø được thể hiện là null
- Hiệu quả hơn



11

- Bảng băm là cấu trúc dữ liệu để cài đặt kiểu dữ liệu từ điển.
- Kiểu dữ liệu từ điển là một tập các đối tượng dữ liệu được thao tác với 3 phép toán cơ bản:
  - Tìm kiếm
  - Chèn
  - Loại bỏ (xóa)
- Dạng đơn giản nhất của bảng băm là một mảng các mẫu tin.
- Mỗi mẫu tin có một trường dữ liệu đặc biệt gọi là khóa (key)

- **Bảng băm** là một cấu trúc dữ liệu sử dụng hàm băm (Hash Function) để ánh xạ các khóa (ví dụ như tên của một người) thành các địa chỉ (chỉ số của mảng), mà nội dung tại địa chỉ này là dữ liệu cần lưu trữ (chẳng hạn là tên người đó, hay số điện thoại của họ).

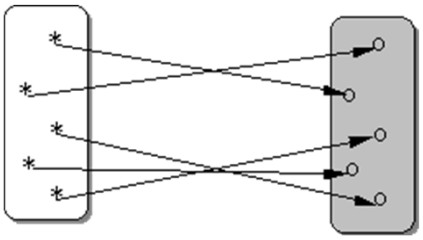
  Do đó, bảng băm là một mảng kết hợp.

12

2. Cấu trúc bảng băm:

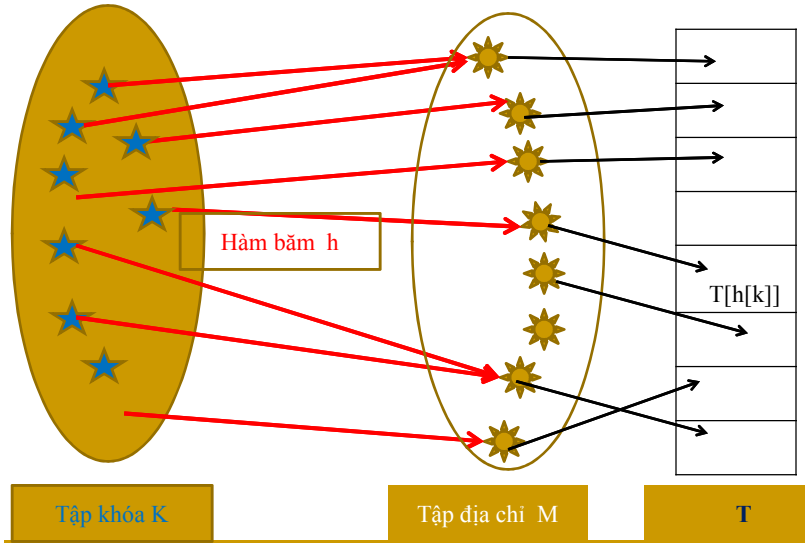
A. Mô tả dữ liệu:

- K: tập các khoá (set of keys)
- M: tập các địa chỉ (set of addresses).
- h(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

Lược đồ bảng băm T gồm Tập khóa K, tập địa chỉ M và hàm băm h



- Trong lược đồ, một dữ liệu có khoá là k, nếu tính địa chỉ của k theo một hàm nào đó ta thu được chỉ số i, 0 <= i < SIZE gọi là địa chỉ i, thì dữ liệu có khóa k được lưu trong thành phần mảng T là T[i].
- Một hàm biến đổi mỗi khoá k thành một địa chỉ, hàm này được gọi là hàm băm (hash function).
- Phương pháp lưu tập dữ liệu theo lược đồ trên được gọi là phương pháp băm (hashing).
- Trong lược đồ, mảng T được gọi là bảng băm (hash table).

3. Các phép toán trên bảng băm:

- Khởi tạo (Initialize): Khởi tạo bảng băm, cấp phát vùng nhớ, quy định số phần tử của bảng (kích thước của bảng).
- Kiểm tra rỗng (Empty): Kiểm tra liệu bảng băm có rỗng hay không.
- Lấy kích thước bảng băm (Size): Lấy số phần tử hiện thời có trong bảng băm.
- Tìm kiếm (Search): Tìm một phần tử theo một khoá k cho trước.
- Thêm mới một phần tử (Insert): Chèn thêm một phần tử vào một vị trí trống của bảng băm.
- Xoá (Delete): Loại bỏ một phần tử khỏi bảng băm.
- Sao chép (Copy): Tạo một bảng băm mới trên cơ sở một bảng băm đã có.
- Duyệt (Traverse): Duyệt các phần tử của bảng theo một thứ tự nhất định.

(Các phép toán trên được cài đặt khác nhau theo từng loại bảng băm.)

4. Hàm băm (Hash Function):

K là tập các khóa của nút. M = {0,..., SIZE -1}, SIZE là số nguyên dương

$$h : K \to M$$
$$k \mapsto h(k) \in M$$

h được gọi là hàm băm,

h(k) được gọi là giá trị băm của khoá k, là địa chỉ trong M.

Khóa k có thể có giá trị số hay chuỗi.

- Tiêu chuẩn để đánh giá hàm băm:
  - Tính toán nhanh
  - Các khóa được phân đều trong bảng
  - Ít xảy ra đụng độ.
  - Xử lý được các loại khóa có kiểu dữ liệu khác nhau
- Nếu băm với khóa không phải là số nguyên, phương pháp giải quyết như sau:
  - Tìm cách biến đổi khóa thành số nguyên
  - Sau đó dùng hàm băm chuẩn trên các số nguyên.

5. Vấn đề xung đột:

- xung đột (collision) là hiện tượng xảy ra khi các khóa khác nhau nhưng băm cùng một địa chỉ, tức là xảy trường hợp h không phải là đơn ánh:

  $k_1 \neq k_2$ và $h(k_1) = h(k_2)$

- Vấn đề đặt ra:
  - Giải quyết sự xung đột ?
  - Xây dựng hàm băm với các tiêu chuẩn: tính nhanh và dễ dàng địa chỉ ứng với mỗi khoá, đảm bảo ít xảy ra xung đột.

6. Giải quyết xung đột

Hai phương pháp giải quyết xung đột:

- Phương pháp định địa chỉ mở
- Phương pháp tạo dây chuyền

- Mỗi phương pháp có thể có nhiều cách giải quyết khác nhau.
- Mỗi phương pháp giải quyết xung đột sẽ xác định loại bảng băm tương ứng.

7. Phân loại bảng băm:

Có các loại:

- Loại bảng băm đơn giản.
- Loại bảng băm xác định bởi phương pháp giải quyết xung đột.

a. Loại bảng băm đơn giản:

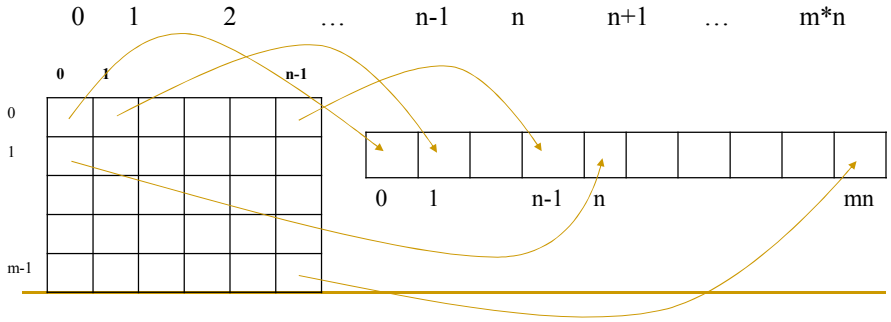
Loại bảng băm đơn giản cài đặt bằng danh sách kề. Để truy xuất một nút trên bảng băm ta chỉ cần biết 2 khóa của nó tương ứng với hàng i, cột j của nút trên bảng.

- Bảng chữ nhật (m hàng n cột)

  Một phần tử của bảng được xác định bởi khoá i ở hàng i và khoá j ở cột j. Tổng quát vị trí của phần tử này có thể xác định qua công thức:

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

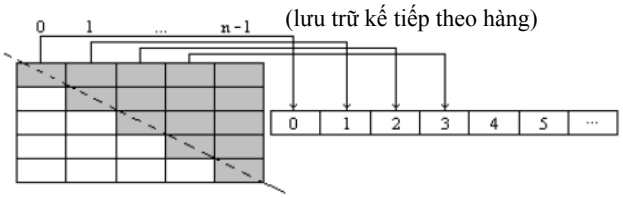
  Bảng băm hình chữ nhật được mô tả bởi một danh sách kề (lưu trữ kế tiếp theo hàng):



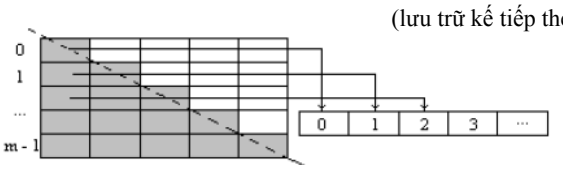
- Bảng băm tam giác trên n cột - Bảng băm tam giác dưới m hàng:
  Mỗi phần tử trên bảng tam giác tương ứng với hàng i, cột j (i ≠ j) và địa chỉ của nó được xác định qua hàm băm:

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

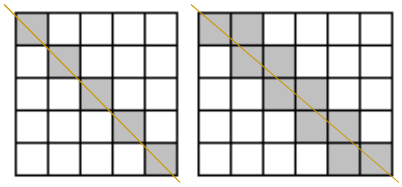
- **Bảng băm tam giác trên n cột:**



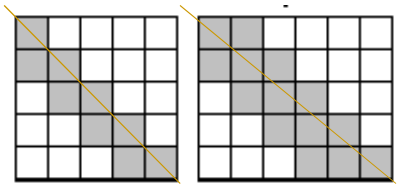
- **Bảng băm tam giác dưới m hàng:**



- *Bảng băm đường chéo:*
  Loại bảng băm đường chéo có dạng sau:



i = j i = j hoặc i = j-1



i = j hoặc i = j+1 i = j hoặc i = j±1

b. Loại bảng băm xác định bởi phương pháp giải quyết xung đột

- Phương pháp định địa chỉ mở: xác định bảng băm định địa chỉ mở
- Phương pháp tạo dây chuyền: xác định bảng băm tạo dây chuyền

II. Các hàm băm:

- Hàm băm được chia làm hai dạng chính:
  - Dạng bảng tra.
  - Dạng công thức.

- Hàm băm dạng bảng tra:

  Giả sử có bảng tra có khoá là bộ chữ cái tiếng Anh. Bảng có 26 26 địa chỉ có giá trị từ 0..25. Khoá *a* ứng với địa chỉ 0, khoá *b* ứng với địa chỉ 1… khoá *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	/	/

- **Hàm băm dạng công thức**:

  Hàm băm thường có dạng công thức tổng quát là h(*k*), trong đó *k* là khoá. Hàm băm dạng này rất đa dạng và không bị ràng buộc bởi bất cứ tiêu chuẩn nào.

  Hàm băm dạng công thức thường dùng với khóa là số nguyên hay xâu ký tự

1. Phép chia lấy dư:

$$h: N \rightarrow M = \{0,..., 9\}$$
$$k \mid\rightarrow h(k) = k \% 10$$

Hàm băm này băm các số nguyên dương thành 10 địa chỉ từ 0 đến 9. Các khóa có hàng đơn vị bằng 0 băm vào địa chỉ 0, có hàng đơn vị là 1 băm vào địa chỉ 1,...

- Tổng quát hơn, với SIZE là số nguyên dương, hàm băm:

$$h: N \rightarrow M = \{0,..., SIZE -1\}$$
$$k \mid\rightarrow h(k) = k \% SIZE$$

sẽ băm các số nguyên dương vào SIZE địa chỉ:

0, 1,..., SIZE-1

- Ưu điểm: đơn giản.
- Hạn chế: xảy ra xung đột nhiều.

- *Để hạn chế xung đột là vấn đề chọn giá trị SIZE*
  - *SIZE = 2^n (không tốt), h(k) = k mod 2^n sẽ chọn cùng n bits cuối của k*
  - *SIZE là nguyên tố (tốt). Thông thường SIZE được chọn là số nguyên tố gần với 2^n. Chẳng hạn bảng ~4000 mục, chọn SIZE = 4093*
  - *Tốt hơn là số nguyên tố có dạng đặc biệt, chẳng hạn dạng 4m+3, m $\in N$.*

Ví dụ, có thể chọn SIZE = 811,

(vì 811 là số nguyên tố và 811 = 4. 202 + 3.)

Hàm băm này được cài đặt trong C++ như sau:

```
unsigned int hash(int k, int SIZE)
{
    return k % SIZE;
}
```

2. Hàm nhân

Tính giá trị băm của khoá k như sau

- Đầu tiên, tính α k, với α một hằng số thực: $0 < \alpha < 1$.
- Tính h(k):

$$h(k) = [(\alpha k)SIZE]$$

với [x] là phần nguyên của số thực x

- *Chọn α và SIZE*
  - *SIZE* thường A phụ thuộc vào đặc trưng của dữ liệu.
  - Theo Knuth chọn $\alpha = \Phi^{-1} = 1/2(SQRT(5)-1) = 0{,}6180399$ được xem là tốt.

3. Hàm băm sử dụng phép nghịch đảo

Hàm băm có dạng:

$$h(k) = \left(\frac{A}{B * k + C} \right) \% SIZE$$

Trong đó: k là khoá, SIZE là kích thước bảng, A, B, C là các hằng số.

4. Hàm băm với các giá trị khoá là xâu ký tự

- Có thể áp dụng phương pháp chuyển đổi một số trong hệ đếm chuyển đổi một xâu ký tự thành một số nguyên.

Ví dụ:

"NOTE" → 'N'.128^3 + 'O'.128^2 + 'T'.128+'E' = 78.128^3 + 79.128^2 +84.128 + 69

- Thông thường một xâu ký tự được tạo thành từ 26 chữ cái và 10 chữ số → thay 128 bởi 37 và tính số nguyên ứng với xâu ký tự theo luật Horner:

"NOTE" → 78.37^3 + 79.37^2 +84.37+ 69 = ((78.37 + 79).37 + 84).37 +69

- Sau khi chuyển đổi xâu ký tự thành số nguyên bằng phương pháp trên, chúng ta sẽ áp dụng phép chia để tính giá trị băm

Hàm băm các xâu ký tự được cài đặt như sau:

```
unsigned   int   hash(const string &k, int SIZE)
{
   unsigned  int  value = 0;
   for (int i=0; i< k.length(); i++)
         value = 37 * value + k[i];
   return    value % SIZE;
}
```

5. Phép băm phổ quát (Universal Hashing)

- Ta thấy có nhiều loại hàm băm khác nhau, nhưng cần phải chọn được một hàm băm thích hợp để hạn chế hiện tượng xung đột giữa các khoá. Giải pháp đưa ra là sử dụng hàm băm phổ quát.
- Băm phổ quát nghĩa là chúng ta chọn ngẫu nhiên một hàm băm h trong một tập các hàm băm H khi thuật toán bắt đầu. Hàm băm được chọn phải đảm bảo:
  - Có tính chất ngẫu nhiên.
  - Đảm bảo các khoá ít xảy ra xung đột.
- Gọi H là tập hữu hạn các hàm băm ánh xạ một tập các khoá K thành các giá trị nằm trong khoảng {0, 1, …, SIZE - 1}. H gọi là phổ quát nếu:
  - Mỗi cặp khoá riêng biệt x, y $\in$K số các hàm băm h $\in$ H cho kết quả h(x) = h(y) là |H| / SIZE.
  - Nói cách khác với mỗi hàm băm ngẫu nhiên từ H khả năng xung đột giữa x và y ($x \neq y$) chính xác là 1/SIZE

Tập ***H*** sẽ được xây dựng như sau:

- Chọn kích thước bảng *SIZE* là một số nguyên tố.
- Phân tích khoá x thành $r + 1$ byte để x có dạng:

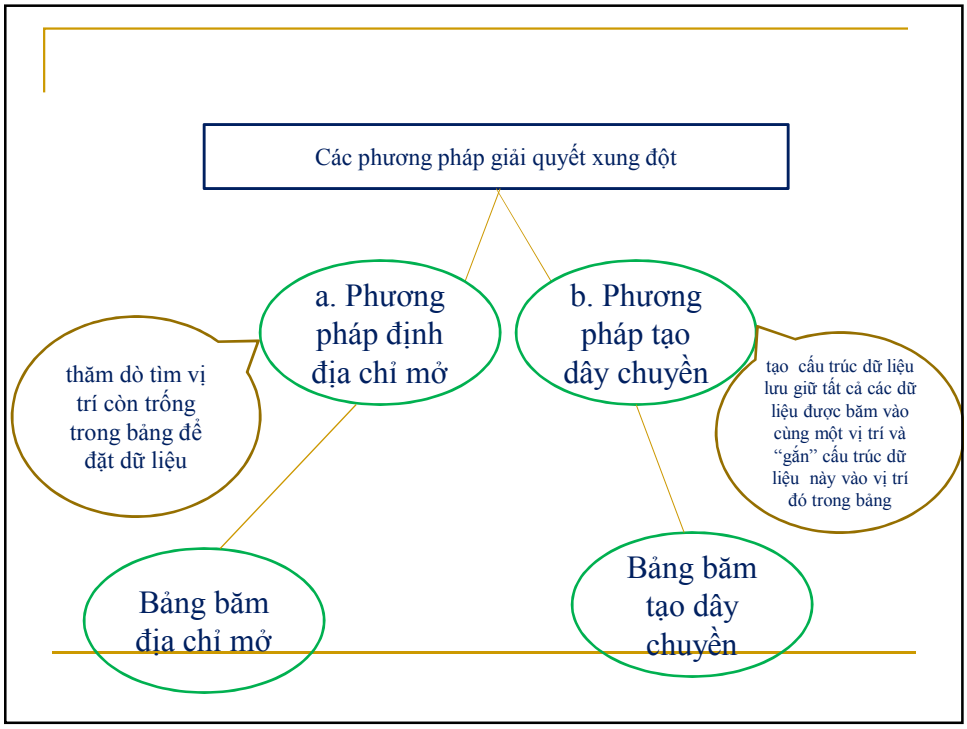
  $$x = \{x_1, x_2, ..., x_r\}.$$

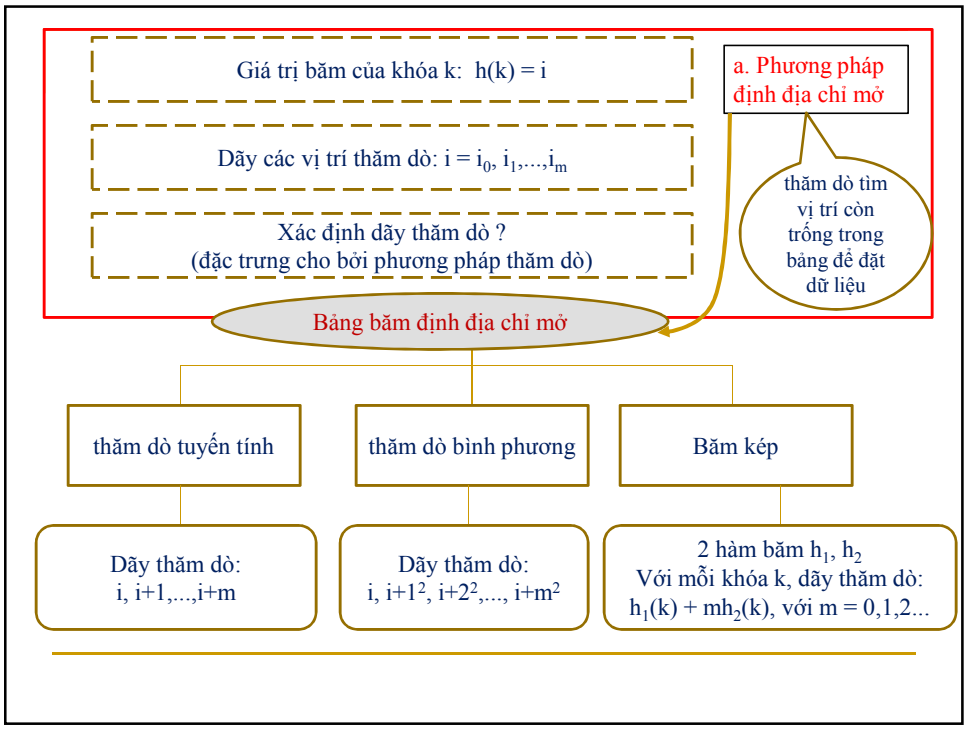
- Giá trị lớn nhất của chuỗi sau khi phân tích < *SIZE*.
- Gọi $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ biểu thị cho một chuỗi $r + 1$ phần tử được chọn trong khoảng $\{0, 1, \ldots, SIZE - 1\}$.
- Hàm băm $h \in \boldsymbol{H}$ tương ứng được định nghĩa như sau:

  $$h_\alpha(x) = \sum_{i=0}^{SIZE} \alpha_i x_i \,\%SIZE$$

  Theo định nghĩa ở trên ***H*** có $SIZE^{r+1}$ phần tử.

III. Các phương pháp giải quyết xung đột


Các phương pháp giải quyết xung đột
a. Phương pháp định địa chỉ mở
b. Phương pháp tạo dây chuyền
thăm dò tìm vị trí còn trống trong bảng để đặt dữ liệu
tạo cấu trúc dữ liệu lưu giữ tất cả các dữ liệu được băm vào cùng một vị trí và "gắn" cấu trúc dữ liệu này vào vị trí đó trong bảng
Bảng băm địa chỉ mở
Bảng băm tạo dây chuyền


Giá trị băm của khóa k: h(k) = i
Dãy các vị trí thăm dò: i = i0, i1,...,im
Xác định dãy thăm dò ?
(đặc trưng cho bởi phương pháp thăm dò)
a. Phương pháp định địa chỉ mở
thăm dò tìm vị trí còn trống trong bảng để đặt dữ liệu
Bảng băm định địa chỉ mở
thăm dò tuyến tính
thăm dò bình phương
Băm kép
Dãy thăm dò: i, i+1,...,i+m
Dãy thăm dò: i, i+1², i+2²,..., i+m²
2 hàm băm h1, h2
Với mỗi khóa k, dãy thăm dò:
h1(k) + mh2(k), với m = 0,1,2...

Cài đặt bảng băm định địa chỉ mở (với các phương pháp thăm dò):

```
//Khai báo cấu trúc một nút của bảng băm
typedefstruct tagNODE
{       //DataType  Data;
        int key;//khoa cua nut tren bam
}NODE;
//Khai báo bảng băm có SIZE nút
NODE hashtable[SIZE];
int N; Số nút hiện có trong bảng
```

- **Kiểm tra bảng băm có rỗng?**

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

- **Kiểm tra bảng băm có đầy?**

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

- **Khởi động bảng băm định địa chỉ mở (với các phương pháp thăm dò):**

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

- **Chọn hàm băm (với các phương pháp thăm dò):**

```
//Chẳng hạn chọn hàm:
int HashFunc(int key)
{
          return(key % SIZE);
}
```

Các thao tác khác: tìm kiếm, chèn, hủy phụ thuộc vào các phương pháp thăm dò

a1. Phương pháp thăm dò tuyến tính (Linear Probing Method)

Giá trị băm của khóa k: $h(k) = i \in \{0, 1, ..., SIZE - 1\}$

Dãy các vị trí thăm dò (tuyến tính): i, i+1,...,i+m

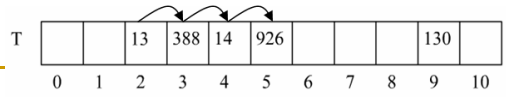
Bảng băm định địa chỉ mở với pp thăm dò tuyến tính

Tìm kiếm khóa k	Chèn khóa k	hủy khóa k
Tìm theo dãy thăm dò: i, i+1,.., i+m - Tìm thấy, xong - Dữ liệu không có trong mảng khi tìm gặp vị trí trống.	Nếu địa chỉ i bị chiếm, tìm đến i+1,i+2,...cho tới khi gặp vị trí trống để chèn	- Tìm thấy vị trí p. - đặt vị trí p ở trang thái delete

Phép toán tìm kiếm dữ liệu có khóa k

- Băm khoá k: h(k)=i.
- Tìm theo dãy thăm dò: i, i+1,.., i+m ***(Không phải tìm hết mảng)***
- Nếu gặp vị trí trống trong dãy thăm dò → dữ liệu cần tìm không có trong mảng.
- Ngược lại→ tìm thấy

- Ví dụ trên: khoá là 47 → h(47) = 47%11 = 3, lần lượt xem xét các vị trí 3, 4, 5, đến vị trí 6 (là vị trí trống) →47 không có trong mảng




```
/* Tim kiem nut co khoa k tren bang bam: neu khong tim thay ham nay tra ve
M (SIZE), neu tim thay ham nay tra ve đia chi tim thay */

int Search (int k)
{
    int 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)% SIZE)
        i = i + 1;
        if(i >= SIZE)
            i = i - SIZE;
    }
    if(hashtable[i].key == k)//tim thay
        return i;
    else//khong tim thay
        return SIZE;
}
```

Chèn dữ liệu với khóa k:

- Nếu i không xung đột: T[h[k]= t[i] = k
- Nếu địa chỉ i bị chiếm, tìm đến các vị trí kế tiếp i+1,i+2,...cho tới khi gặp vị trí trống. Chèn dữ liệu vào vị trí trống này.

Ví dụ:

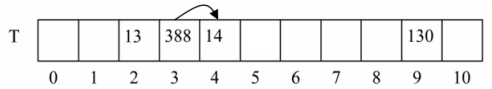
- SIZE = 11.
- T khởi tạo là rỗng.

T[i]											
i:	0	1	2	3	4	5	6	7	8	9	10

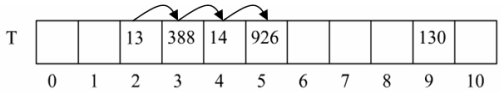
- Dữ liệu cần chèn vào T có các khoá là:
  388, 130, 13, 14, 926.
- Với hàm băm là hàm chia cho SIZE lấy dư.
- Băm khóa k, ta có: i = h(k) = k%SIZE

- h(388) = 3 → T[3] = 388; h(130) = 9→T[9]=130; h(13) = 2→T[2] = 13.

- h(14) = 3→ xung đột (T[3]=388), vị trí tiếp theo trống là 4→T[4] = 14.



- h(926) = 2, tìm đến các vị trí tiếp theo 3, 4, 5 và 926 được đặt vào T[5]. Kết quả là chúng ta nhận được mảng T:



/* Chèn nút có khóa k vào bảng băm, kích thước bảng băm là M */

```
int Insert (int k)
{
    int i, j;
    if(Isfull())
        return SIZE; //Bang day
    i = HashFunc(k);
    while(hashtable[i].key != NULLKEY)
    {
        i = i+1;
        if(i >= SIZE)
            i = i-SIZE;
    }
    hashtable[i].key == k;
     N = N+1; //So nut hien co cua bang bam
    return i;
}
```

Phép toán loại (hủy) dữ liệu có khóa k

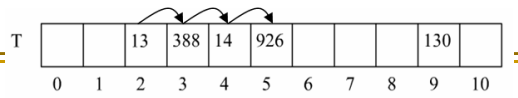
- Áp dụng thao tác tìm kiếm, giả sử tìm thấy tại vị trí p.
- Đặt vị trí p là vị trí trống

Vấn đề:

- Loại 388 → T[3] = trống
- Yêu cầu: tìm 926 → h (926) = 2 và T[2] không chứa 926, tìm đến vị trí 3 là trống →kết luận 926 không có trong mảng→**Sai**

Giải quyết:

- Đánh dấu trạng thái cho mỗi vị trí: trống (EMPTY), loại bỏ (DELETED), chứa dữ liệu (ACTIVE).
- Khi chèn dữ liệu mới, chúng ta có thể đặt nó vào vị trí đã loại bỏ.



Sửa lại thao tác Chèn dữ liệu với khóa k như sau:

Nếu địa chỉ i bị chiếm, tìm đến các vị trí kế tiếp i+1,i+2,...cho tới khi gặp vị trí trống (Empty) hay loại bỏ (Delete). Chèn dữ liệu vào vị trí này.

- **Ưu điểm:**

  Phép toán chèn luôn luôn thực hiện được, trừ khi mảng đầy
- **Hạn chế:**

  Dữ liệu tập trung thành từng đoạn → các phép toán kém hiệu quả, chẳng hạn nếu i = h(k) ở đầu một đoạn, để tìm dữ liệu với khoá k chúng ta cần xem xét cả một đoạn dài.

a2. Phương pháp thăm dò bình phương (Quadratic Probing Method)

Giá trị băm của khóa k: $h(k) = i \in \{0, 1,..., SIZE -1\}$

$i, (i+1^2)\%SIZE, (i+2^2)\%SIZE,..., (i+m^2)\%SIZE,...$

Bảng băm định địa chỉ mở với pp thăm dò bình phương

Ví dụ:

Nếu cỡ của mảng SIZE = 11, và i = h(k) = 3, thì thăm dò bình phương cho phép ta tìm đến các địa chỉ:

3, 4, 7, 1, 8 và 6.

```
//Insert:them nut co khoa k vao bang bam

int Insert(int k)
{
    if (IsFull())
    {
        cout<<"\n Bang bam bi day khong them nut co khoa "<<k<<" duoc!";
        return;
    }
    int j = 1;
    int i = HashFunc(k);
    while(hashtable[i].key != NULLKEY)
    {
        //Bam lai (theo phuong phap do binh phuong)
        i = i +j*j; j++;
        if (i >SIZE)
            i = i%SIZE;
    }
    hashtable[i].key = k;
    N = N+1;
    return i;
}
```

- **Ưu điểm:** tránh được sự tích tụ dữ liệu thành từng đoạn, tránh được sự tìm kiếm tuần tự trong các đoạn.
- **Hạn chế:** không cho phép ta tìm đến tất cả các vị trí trong mảng → phép toán chèn có thể không thực hiện được, mặc dầu trong mảng vẫn còn các vị trí không chứa dữ.

a3. Phương pháp băm kép (Double Hashing Method)

Sử dụng 2 hàm băm h_1, h_2:
- Hàm băm h_1: xác định vị trí thăm dò đầu tiên.
- Hàm băm h_2: xác định bước thăm dò

- Dãy thăm dò:
  $h_1(k) + mh_2(k)$, m = 0,1,2,...
  $h_2(k) \neq 0$, $\forall k$

Bảng băm định địa chỉ mở với pp băm kép

b. Phương pháp tạo dây chuyền (Kết nối trực tiếp) (Direct Chaining Method)

Bảng băm với phương pháp tạo dây chuyền

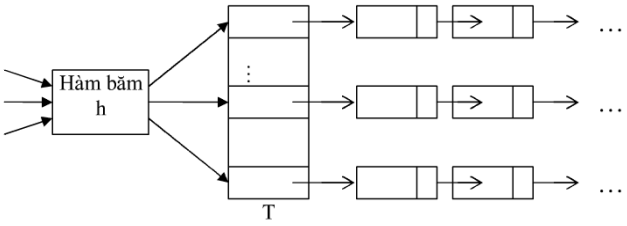
Giá trị băm của khóa k: $h(k) = i \in \{0, 1,..., SIZE -1\}$

Giải quyết xung đột: các khóa xung đột tại địa chỉ i sẽ lưu trữ trong 1 danh sách liên kết

Giá trị băm của khóa k: h(k) = i ∈ {0, 1,..., SIZE -1}

T(h(k)) = T(i), i ∈ {0, 1,..., SIZE -1}

Mỗi T[i] sẽ chứa con trỏ trỏ tới đầu một DSLK
(DSLK này chứa các khóa xung đột tại địa chỉ i)



- **Ưu điểm:** số dữ liệu được lưu không phụ thuộc vào cỡ của mảng. Việc tìm kiếm hoặc loại phụ thuộc vào độ dài của DSLK.
- **Hạn chế:** chỉ hạn chế bởi bộ nhớ cấp phát động cho các dây chuyền.

Cài đặt bảng băm dây chuyền

Trong mục này chúng ta sẽ cài đặt bảng băm dây chuyền.

```
// Cấu trúc tế bào trong dây chuyền.
struct      Cell
{
    Item    data;
    Cell*   next;
};
Cell*   T[SIZE];// Mảng các con trỏ trỏ  đầu các dây chuyền
// Các phép toán từ điển:
bool   Search(int k, Item & I) ;
void    Insert(const Item & object, bool & Suc);
void    Delete(int   k);
```


Để khởi tạo ra bảng băm rỗng, chúng ta chỉ cần đặt các thành phần trong mảng T là con trỏ NULL.

Hàm kiến tạo mặc định như sau:

```
for (int i= 0 ; i< SIZE ; i++)
T[i] = NULL;
```

Các hàm tìm kiếm, xen, loại được cài đặt rất đơn giản, sau khi băm chúng ta chỉ cần áp dụng các kỹ thuật tìm kiếm, xen, loại trên các DSLK. Các hàm Search, Insert và Delete được xác định dưới đây:

```
bool Search(int k, Item & I)
{
    int  i = Hash(k);
    Cell*  P = T[i];
    while (P ! = NULL)
        if (P→data.key = = k)
        {
                I = P→data;
                return   true;
        }
        else
                P= P→next;
    return    false;
}
```

```
void   Insert(const Item &  object, bool &  Suc)
{
    int   i = Hash(k);
    Cell*   P = new Cell;
    if (P != NULL)
    {
        P→data = object;
        P→next = T[i];
        T[i] = P;  //Xen vào đầu dây chuyền.
        Suc = true;
    }
    else
        Suc  = false;
}
```

```
void   Delete(int  k)
{
    int   i = Hash(k);
    Cell*   P;
    If (T[i] != NULL)
    If (T[i] →data.key = = k)
    {
          P = T[i];
          T[i] = T[i] →next;
          delete   P;
    }
    else
```



```
    {
          P = T[i];
          Cell*  Q = P →next;
          while (Q != NULL)
                if (Q →data.key = = k)
                {
                      P →next = Q →next;
                      delete   Q;
                      Q = NULL;
                }
                else
                {
                      P  =  Q;
                      Q  =  Q →next;
                }
    }
}
```

IV. Ứng dụng từ điển

- Sử dụng cấu trúc dữ liệu bảng băm theo phương pháp tạo dây chuyền (kết nối trực tiếp) xây dựng chương trình từ điển Anh – Việt

- Cài đặt cấu trúc dữ liệu bảng băm theo phương pháp dây chuyền:

```
#define SIZE 26
struct tagNODE
{
        char word [10] ;
        char mean[50];
        tagNODE *Next;
};
typedef tagNODE NODE;
typedep NODE, *LPNODE;
LPNODE bucket[SIZE];
```

```
//Chọn hàm băm
int Hashfunc(char word [])
{
        char ch = toupper(word[0]);
        return ((ch-65) % SIZE);
}
```

```
//Tạo nút
LPNODE GetNode(char word[], char mean[])
{
        LPNODE p =new NODE;
        strcpy(p->word, word);
        strcpy(p->mean, mean);
        p->Next = NULL;
        return p;
}
//_____________________________
// Khởi tạo thùng bucket
void Initialize ()
{
        for (int b=0; b<SIZE; b++)
                bucket[b] = NULL;
}
```

```
//  Thêm một nút i vào vào thùng bucket
void Insert (LPNODE p)
{
        int i=Hashfunc(p->word);
        p->Next=bucket[i];
        bucket[i]=p;
}
//_____________________________________________
```

```
/* Hàm tạo từ điển */
void MakeDictionary()
{
        LPNODE p;
        char word[10];
        char mean[50];
        do
        {
                _fflush(stdin);
                cout<<"\n Nhap tu can dua vao tu dien:";
                _gets(word);
                if (!strcmp(word,""))
                        break;
                _fflush(stdin);
                cout<<"\nNhap nghia cua tu: ";
                _gets(mean);
                p = GetNode(word,mean);
                Insert(p);
         }
         while (1);
}
```

```
// Hàm tìm một từ trong từ điển
LPNODE Find(char word[])
{
        int done = 1;
        int i = Hashfunc(word);
        LPNODE 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 tra nghĩa của từ trong từ điển
void FindWord()
{
        char word[10];
        cout<<"\nNhap tu: ";
        _fflush(stdin);
        gets(word);
        LPNODE p = Find(word);
        if(p == NULL)
                cout<<"\nKhong co trong tu điên";
        else
                cout<<"\n Co tu:"<<p->word<<"\nNghia la: "<<p->mean;
}
```