

CONFIDENTIAL

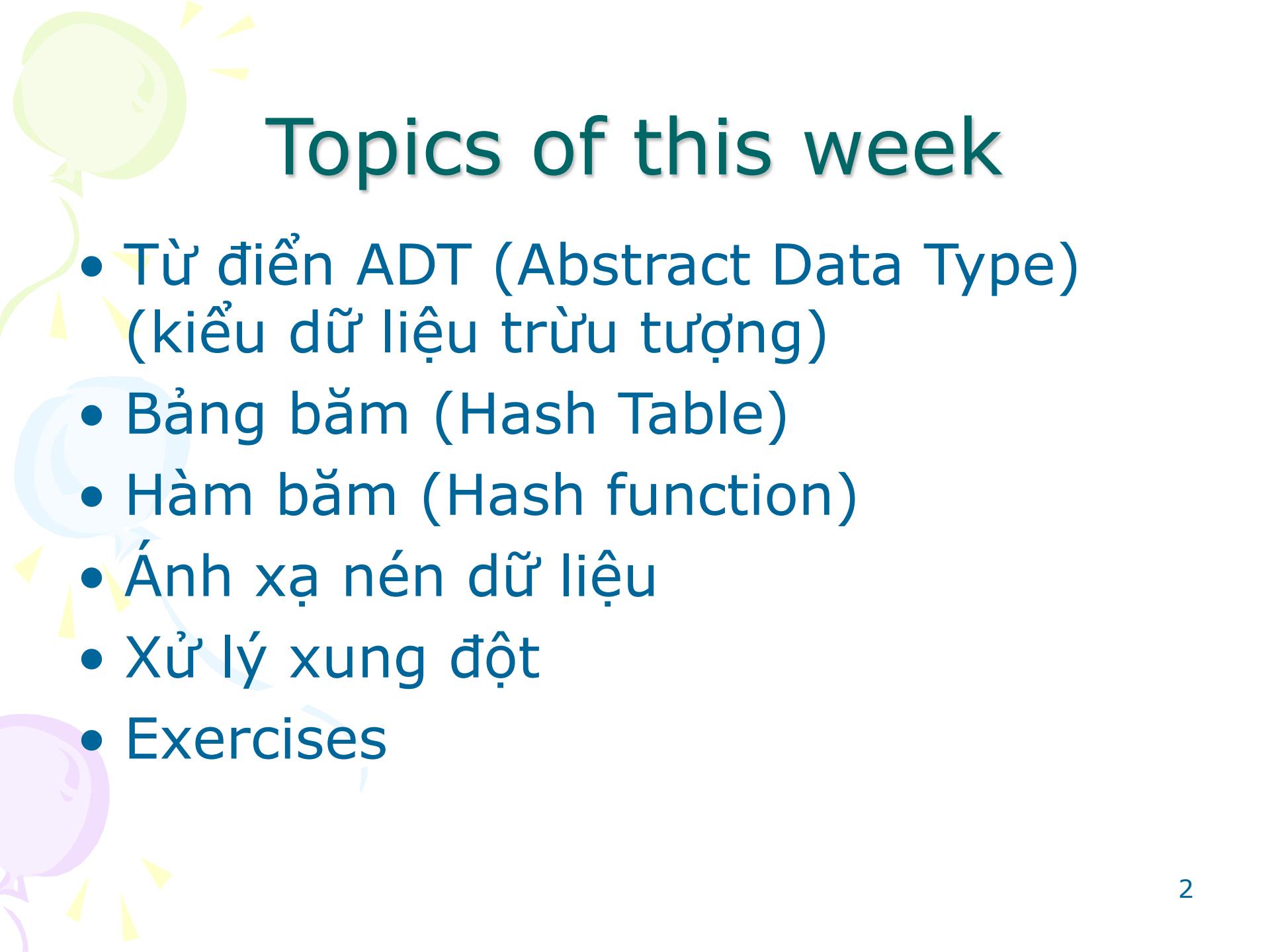
C Programming Basic – week 14

Mapping and Hashing

Lecturer :

Do Quoc Huy

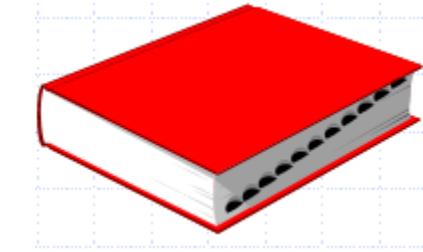
**Dept of Computer Science
Hanoi University of Technology**



Topics of this week

- Từ điển ADT (Abstract Data Type)
(kiểu dữ liệu trừu tượng)
- Bảng băm (Hash Table)
- Hàm băm (Hash function)
- Ánh xạ nén dữ liệu
- Xử lý xung đột
- Exercises

Từ điển ADT



- Từ điển ADT mô hình 1 tập các phần-tử-khoa (key-element)
- Các thao tác chính của 1 từ điển là tìm kiếm, chèn và xóa các mục
- Cho phép nhiều mục có cùng khóa (key)
- Ứng dụng:
 - address book
 - Xác thực credit card
 - Ánh xạ tên miền (v.d., csci260.net) to thành địa chỉ internet (v.d., 128.148.34.101)

Các thao tác với từ điển ADT

- `findElement(k)`: nếu từ điển có 1 mục với khóa `k`, trả về giá trị của phần tử đó, nếu không, trả về phần tử đặc biệt `NO_SUCH_KEY`
- `insertItem(k, o)`: chèn mục `(k, o)` vào từ điển
- `removeElement(k)`: nếu từ điển có 1 mục với khóa `k`, xóa nó khỏi từ điển từ điển và trả về phần tử của nó, nếu không, trả về phần tử đặc biệt `NO_SUCH_KEY`
- `size()`, `isEmpty()`
- `keys()`, `elements()`

Từ điển với khóa-được-đánh-chỉ-số

Key	Value
1	Intro to CS 1
2	Intro to CS 2
5	Theory of Computation
7	Data Structures
9	Digital Logic



A[]

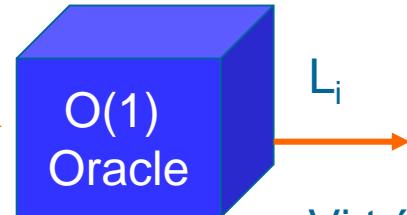
0	
1	Intro to CS 1
2	Intro to CS 2
3	
4	
5	Theory of Computation
6	
7	Data Structures
8	
9	Digital Logic

Khoảng trống-chỉ hiệu quả nếu số các yếu tố trong tập hợp gần với N

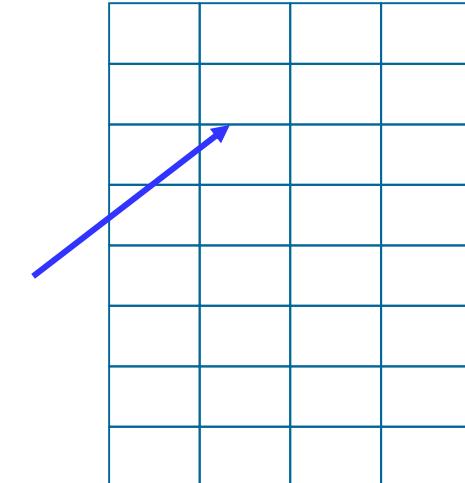
Tìm kiếm không có phép so sánh

- Nếu ta có một thứ như « người chỉ đường/ vật hướng dẫn (Oracle)»- thứ mà có thể lấy giá trị khóa (key) của dữ liệu và tính ra (trong 1 khoảng thời gian xác định) vị trí khóa (key) có thể xuất hiện bên trong tập dữ liệu?

data key K



Vị trí bản ghi
phù hợp bên
trong tập hợp



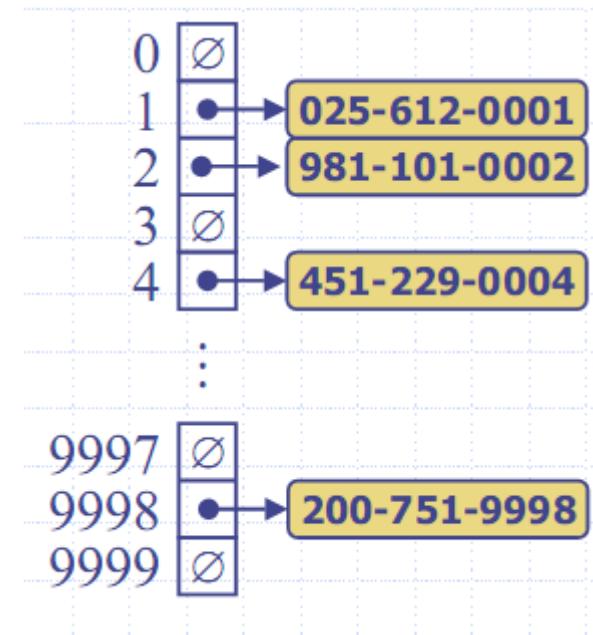
Nếu vật chứa tập hợp hỗ trợ truy cập ngẫu nhiên với chi phí $\Theta(1)$ giống như truy cập vào mảng, thì ta có thể có tổng chi phí của $\Theta(1)$.

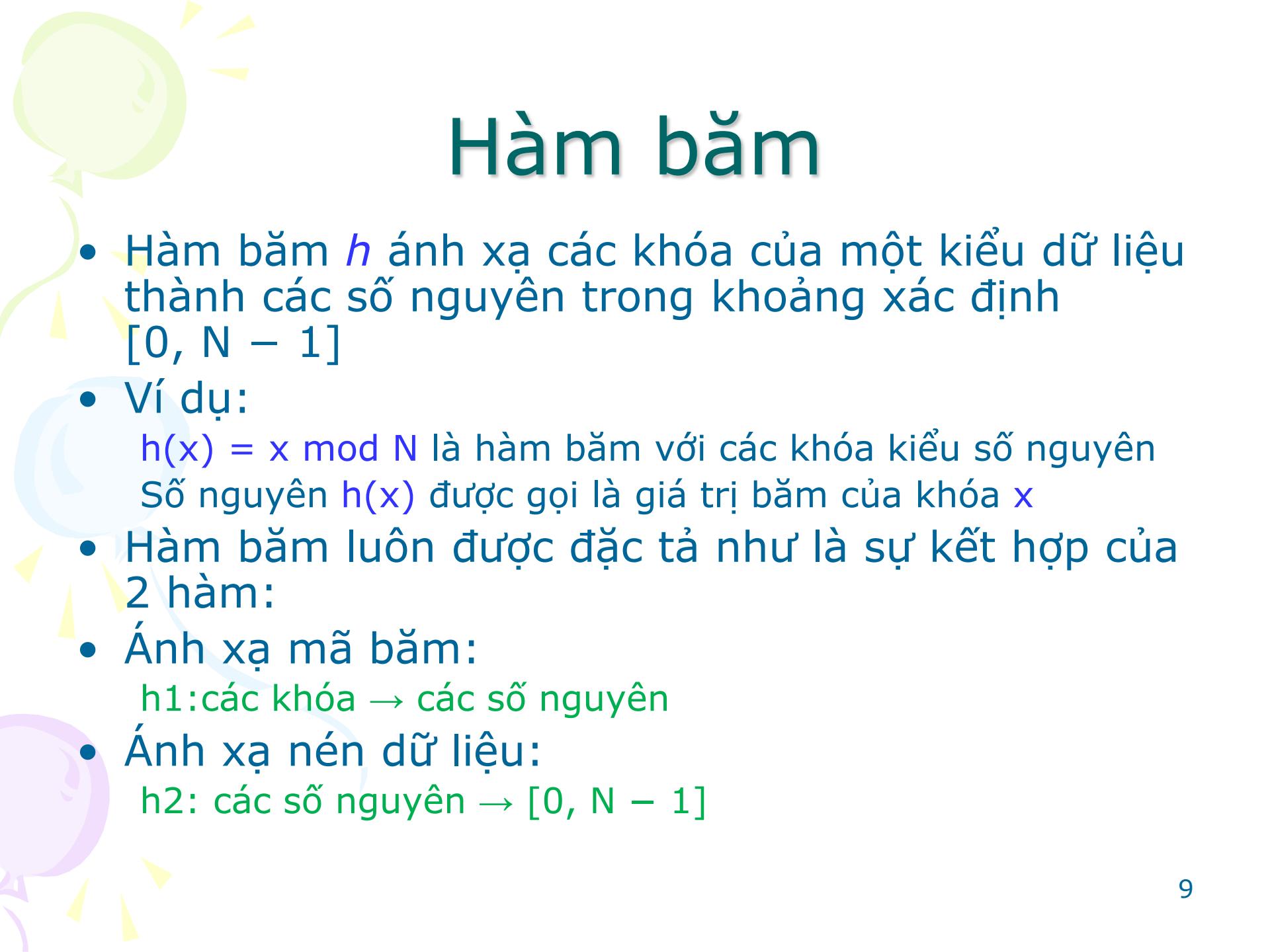
Hàm băm và bảng băm

- Một cách hiệu quả để triển khai từ điển là bảng băm (hash table).
- Sử dụng một mảng (hoặc danh sách) với kích thước N (bảng)
 - Cần rải đều các khóa (key) trên khoảng $[0, N-1]$
 - Sự đụng độ (trùng nhau) xảy ra khi các phần tử có cùng khóa (key)
- Các khóa không phải phải lúc nào cũng là số nguyên hay trong khoảng $[0, N-1]$
- Một **hash table** cho một kiểu khóa sẽ gồm
 - Hàm băm h
 - Mảng (trường hợp này gọi là *bảng*) kích thước N
- Khi triển khai từ điển với bảng băm, mục đích là để lưu mục (k, o) tại chỉ số $i = h(k)$

Ví dụ

- Ta thiết kế một bảng băm cho từ điển lưu giữa các mục (SIN, Name), với SIN (Social Insurance Number) là số nguyên dương 9 chữ số
- Bảng băm sử dụng một mảng có kích thước $N = 10,000$ và hàm băm $h(x) = 4$ chữ số cuối của x





Hàm băm

- Hàm băm h ánh xạ các khóa của một kiểu dữ liệu thành các số nguyên trong khoảng xác định $[0, N - 1]$
- Ví dụ:
 $h(x) = x \text{ mod } N$ là hàm băm với các khóa kiểu số nguyên
Số nguyên $h(x)$ được gọi là giá trị băm của khóa x
- Hàm băm luôn được đặc tả như là sự kết hợp của 2 hàm:
- Ánh xạ mã băm:
 $h1: \text{các khóa} \rightarrow \text{các số nguyên}$
- Ánh xạ nén dữ liệu:
 $h2: \text{các số nguyên} \rightarrow [0, N - 1]$

Ánh xạ mã băm

- Integer cast

- Các bit của khóa được dịch như là các số nguyên
- Thích hợp cho các khóa có độ dài ngắn hơn số bit của kiểu số nguyên
- Ví dụ:
 - 'A' -> 65
 - 'N' -> 78

- Tổng các thành phần (Component Sum)

- Chi các bit của khóa thành nhiều phần với kích thước cố định (v.d. 16 hay 32 bit) và tính tổng các thành phần
- Thích hợp cho các khóa kiểu số có độ dài cố định lớn hơn hoặc bằng số bit của kiểu số nguyên

$$x = (x_1, x_2, \dots, x_{n-1}) \Rightarrow h_1(x) = \sum_{i=0}^{n-1} x_i$$

x_1, x_2, \dots, x_{n-1}
32 bits 32 bits 32 bits

Ánh xạ mã băm (tiếp)

- **Tính giá trị đa thức**

- Chia các bit của khóa thành dãy các phần với kích thước cố định (v.d., 8, 16 hay 32 bit)

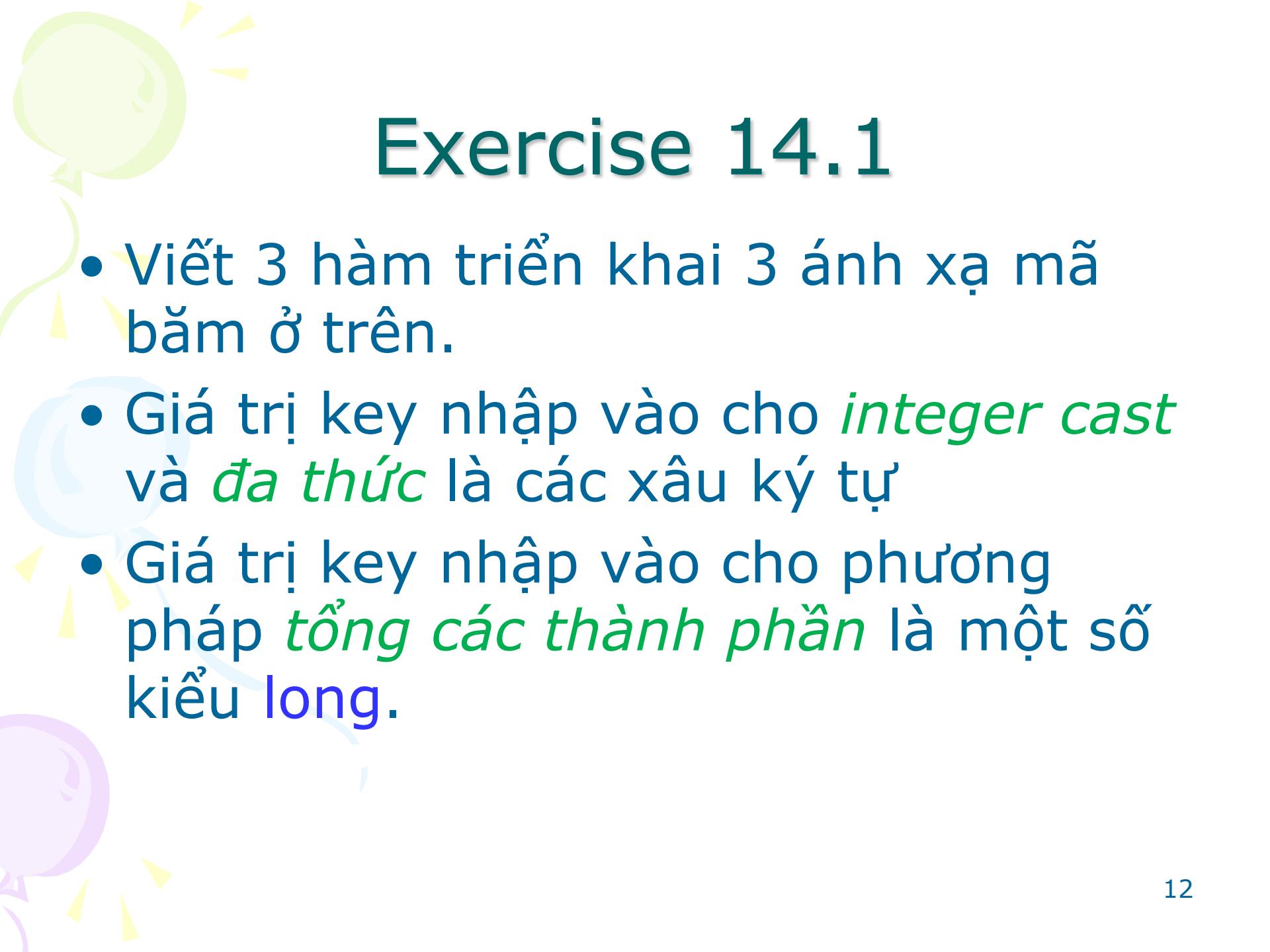
$$a_0 \ a_1 \ \dots \ a_{n-1}$$

- Ta tính đa thức

$$p(z) = a_0 + a_1 z + a_2 z^2 + \dots + a_{n-1} z^{n-1}$$

tại giá trị z *cố định*, bỏ qua việc tràn số

- Đặc biệt thích hợp cho các xâu ký tự (v.d. nếu chọn $z = 33$ sẽ cho tối đa 6 giá trị trùng nhau trên một tập hợp 50,000 từ tiếng Anh)



Exercise 14.1

- Viết 3 hàm triển khai 3 ánh xạ mã băm ở trên.
- Giá trị key nhập vào cho *integer cast* và *đa thức* là các xâu ký tự
- Giá trị key nhập vào cho phương pháp *tổng các thành phần* là một số kiểu long.

Ánh xạ nén dữ liệu

- Kết quả của phần trước cần được thu gọn về 1 giá trị trong khoảng $[0, N-1]$
- Division Method:
 - $h_2(y) = |y| \ mod \ N$
 - K/thước N của bảng băm thường là 1 số nguyên tố

- Multiply, Add and Divide (MAD):
 - $h_2(y) = |ay + b| \ mod \ N$
 - a và b là các số nguyên không âm sao cho $a \ mod \ N \neq 0$
 - Nếu không, mọi số nguyên đều sẽ ánh xạ thành cùng giá trị b

Cài đặt đơn giản của bảng băm

```
#define MAX_CHAR 10
#define TABLE_SIZE 13
typedef struct {
    char key[MAX_CHAR];
    /* các trường khác*/
} element;
element hash_table[TABLE_SIZE];
```

Thuật toán băm với phép chia (Division)

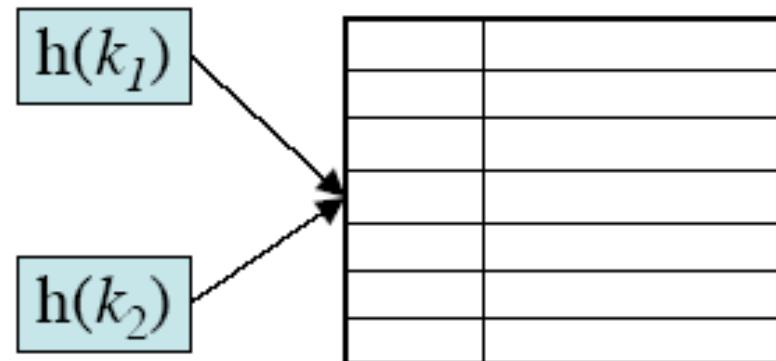
```
void init_table(element ht[])
{
    int i;
    for (i=0; i<TABLE_SIZE; i++)
        ht[i].key[0]=NULL;
}
```

```
int transform(char *key)
{
    int number=0;
    while (*key) number += *key++;
    return number;
}
```

```
int hash(char *key)
{
    return (transform(key)
            % TABLE_SIZE);
}
```

Giải pháp cho xung đột

- Xung đột xảy ra khi $k_1 \neq k_2$ nhưng $h(k_1) = h(k_2)$
- Dẫn đến các thao tác *insertItem()* và *findElement()* phức tạp hơn
- Các chiến lược giải quyết xung đột
 - Đánh địa chỉ đóng (Bảng băm mở) – nghĩa là các ô (slot) không phải là $h(k)$ là “đóng” và không thể s/dụng
 - Đánh địa chỉ mở (Bảng băm đóng)- tìm vị trí khác trong bảng băm

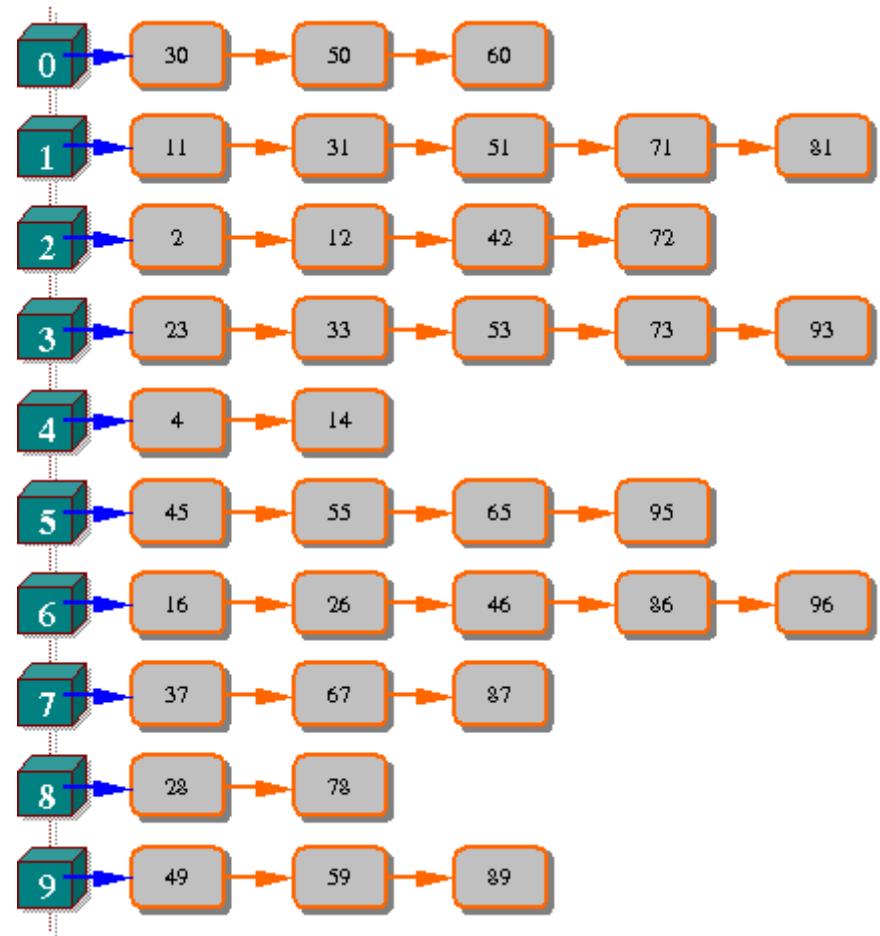


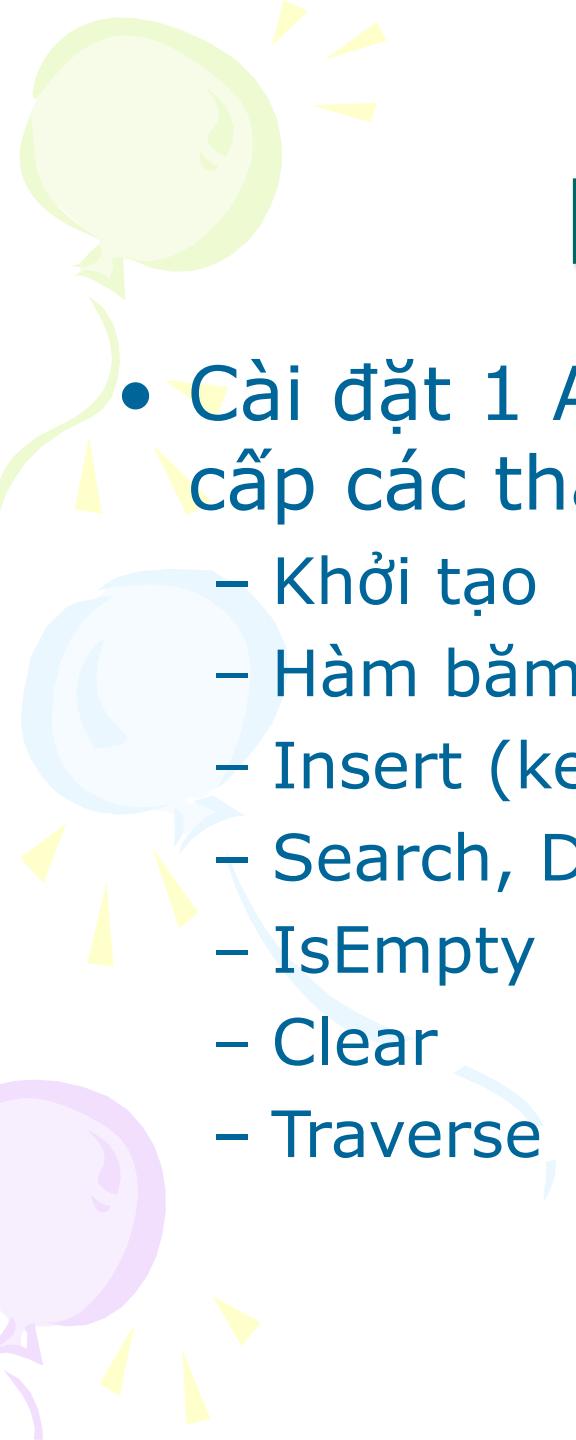
Cấu trúc dữ liệu cho bảng băm

- Bảng băm mở:
 - Phương pháp xích lại (Chaining)
- Bảng băm đóng
 - Thăm dò tuyến tính
 - Thăm dò bậc hai (Quadratic)
 - Băm lại

Cấu trúc dữ liệu cho chaining

- Mảng các con trỏ
- Mỗi con trỏ quản lý 1 d/sách l/kết tương ứng với 1 bucket (địa chỉ).
- Ví dụ này thể hiện bảng băm xích với hàm băm $N \bmod 10$





Exercise 14.2

- Cài đặt 1 ADT cho bảng băm xích, cung cấp các thao tác sau:
 - Khởi tạo
 - Hàm băm
 - Insert (key, element)
 - Search, Delete (key)
 - IsEmpty
 - Clear
 - Traverse

Solution

Khai báo cấu trúc dữ liệu

```
#define B ... // kích thước của bảng băm
typedef ... KeyType; // int
typedef struct Node
{
    KeyType Key;
    // Thêm các trường mới nếu cần
    Node* Next;
};
typedef Node* Position;
typedef Position Dictionary[B];
Dictionary D;
```

Khởi tạo bảng băm

```
void MakeNullSet()  
{  
    int i;  
    for(i=0;i<B;i++)  
        D[i]=NULL;  
}
```

Tìm một phần tử trong bảng băm

```
int Search(KeyType X)    {  
    Position P;  
    int Found=0;  
    P=D[H(X)]; //Tới bucket H(X)  
    //Duyệt danh sách tại bucket H(X)  
    while((P!=NULL) && (!Found))  
        if (P->Key==X) Found=1;  
        else P=P->Next;  
    return Found;  
}
```

Chèn một phần tử

```
void InsertSet(KeyType X)
{
    int Bucket;
    Position P;
    if (!Member(X, D)) {
        Bucket=H(X);
        P=D[Bucket];
        //them một node mới tại D[Bucket]
        D[Bucket] = (Node*)malloc(sizeof(Node));
        D[Bucket] ->Key=X;
        D[Bucket] ->Next=P;
    }
}
```

Xóa một phần tử

```
void DeleteSet(ElementType X){  
    int Bucket, Done;  
    Position P,Q;  
    Bucket=H(X);  
    // Nếu danh sách đã tồn tại  
    if (D[Bucket]!=NULL) {  
        // Nếu X ở đầu danh sách  
        if D[Bucket]->Key==X)  
        {  
            Q=D[Bucket];  
            D[Bucket]=D[Bucket]-  
>Next;  
            free(Q);  
        }  
        else { // Tìm X  
            Done=0;  
            P=D[Bucket];  
            while ((P->Next!=NULL) &&  
                   (!Done))  
                if (P->Next->Key==X)  
                    Done=1;  
                else P=P->Next;  
            if (Done) { // Nếu thấy  
                // Delete P->Next  
                Q=P->Next;  
                P->Next=Q->Next;  
                free(Q);  
            }  
        }  
    }  
}
```

Kiểm tra rỗng

Kiểm tra nếu bucket rỗng

```
int emptybucket (int b) {  
    return (D[b] ==NULL ? 1:0);  
}
```

Kiểm tra nếu bảng rỗng

```
int empty( ) {  
    int b;  
    for (b = 0; b<B;b++)  
        if (D[b] !=NULL)  return 0;  
    return 1;  
}
```

Xóa một bucket

```
void clearbucket (int b){  
    Position p,q;  
    q = NULL;  
    p = D[b];  
    while(p !=NULL){  
        q = p;  
        p=p->next;  
        free (q);  
    }  
    D[b] = NULL;  
}
```

Xóa bảng băm

```
void clear( )
```

```
{
```

```
    int b;
```

```
    for (b = 0; b < B ; b++)
```

```
        clearbucket(b);
```

```
}
```

Duyệt bucket

```
void traversebucket (int b)
{
    Position p;
    p= D[b];
    while (p !=NULL)
    {
        // Giả sử key có kiểu int
        printf("%3d", p->key);
        p= p->next;
    }
}
```

Duyệt bảng

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

Exercise 14-2 Tạo danh sách băm

- Tạo danh bạ điện thoại.
- (1) Đọc 10 danh bạ từ file, lưu vào bảng băm với “địa chỉ e-mail” là khóa(key).
 - Sau đó xác thực là bảng băm đã được tạo.
 - Trong bài tập này, hàm băm có thể luôn trả về cùng giá trị.
- (2) Xây dựng hàm băm một cách hợp lý để việc xung đột xảy ra càng ít càng tốt

Thăm dò tuyển tính (Đánh địa chỉ mở tuyển tính)

- Tính giá trị $f(x)$ cho định danh x
- Kiểm tra các bucket

$$\begin{aligned} & \text{ht}[(f(x)+j)\% \text{TABLE_SIZE}] \\ & 0 \leq j \leq \text{TABLE_SIZE} \end{aligned}$$

- bucket chứa x .
- bucket chứa xâu rỗng
- bucket chứa một xâu không rỗng không phải x
- Trả về cho $\text{ht}[f(x)]$

Linear Probing - example

0	49**
1	58**
2	69**
3	
4	
5	
6	
7	
8	18
9	89

Thăm dò tuyến tính với $f(i) = i$.

Bảng băm ở đây có k/thước $T = 10$, với mục 89, 18, 49, 58, và 69 đã được chèn vào. Hàm băm là $h(key) = key \% 10$.

Trong bài này ta dùng bảng băm có k/thước $T = 10$, mặc dù trong thực tế nên là số nguyên tố.



Exercise 14.3

- Cài đặt bảng băm ADT với phương pháp thăm dò tuyến tính.

Solution: Data structure

```
#define NULLKEY -1
#define M 100 // size of hash table
struct node
{
    int key;
};

//Declare hash table as an array
struct node hashtable[M];
int NODEPTR;
int N = 0;
```

Hàm băm và khởi tạo

```
int hashfunc(int key)
{
    return(key% 10); // or any number
}
```

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

Kiểm tra trạng thái của bảng

```
int full( ) {  
    return (N==M-1 ? 1: 0);  
}
```

```
int empty( ) {  
    return (N==0 ? 1: 0);  
}
```

Search

```
int search(int k) {  
    int i;  
    i=hashfunc(k);  
    while(hashtable[i].key!=k &&  
    hashtable[i].key !=NULLKEY) {  
        //rehash :fi(key)=f(key)+1) % M  
        i=i+1;  
        if(i>=M) i=i-M;  
    }  
    if(hashtable[i].key==k) // found  
        return i;  
    else // not found  
        return M;  
}
```

Insert

```
int insert(int k) {  
    int i, j;  
    if(full()) {  
        printf("\n Hash table is full. Can not insert  
the key %d ", k);  
        return;  
    }  
    i=hashfunc(k);  
    while(hashtable[i].key !=NULLKEY) {  
        // Rehash  
        i++;  
        if(i>M) i= i-M;  
    }  
    hashtable[i].key=k;  
    N=N+1;  
    return i;  
}
```

Remove a key

```
void remove(int i){  
    int j, r, a, cont=1;  
    do {  
        hashtable[i].key = NULLKEY;  
        j = i;  
        do {  
            i=i+1;  
            if(i>=M) i=i-M;  
            if(hashtable[i].key == NULLKEY) cont = 0;  
            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);  
}
```

Thăm dò bậc hai

- Thăm dò tuyển tính có xu hướng bị bó lại
 - Tìm kiếm chậm
- Được t/kẽ để loại bỏ các vân đề bị bó lại chủ yếu của thăm dò tuyển tính
- Xử dụng các hàm xung đột bậc 2 như là $f(i) = i^2$
- Không đảm bảm tìm ra một ô/mục trống nếu bảng đầy hơn 1 nửa trừ khi k/thước bảng là số nguyên tố



Exercise 14.4

- Cài đặt bảng băm ADT với phương pháp thăm dò bậc hai.

Search

```
int search(int k) {  
    int i, d;  
    i = hashfunc(k);  
    d = 1;  
    while(hashtable[i].key!=k && hashtable[i].key  
    !=NULLKEY) {  
        //Quadratic probing  
        i = (i+d*d) % M;  
        d = d+1;  
    }  
    if(hashtable[i].key==k) // found  
        return i;  
    else // not found  
        return M;  
}
```

Insert

```
int insert(int k) {  
    int i, d;  
    if(full()) {  
        printf("\n Hash table is full. Can not insert  
the key %d ",k);  
        return;  
    }  
    i=hashfunc(k); d = 1;  
    while(hashtable[i].key !=NULLKEY) {  
        //Quadratic probing  
        i = (i+d*d) % M;  
        d = d+1;  
    }  
    hashtable[i].key=k;  
    N=N+1;  
    return i;  
}
```

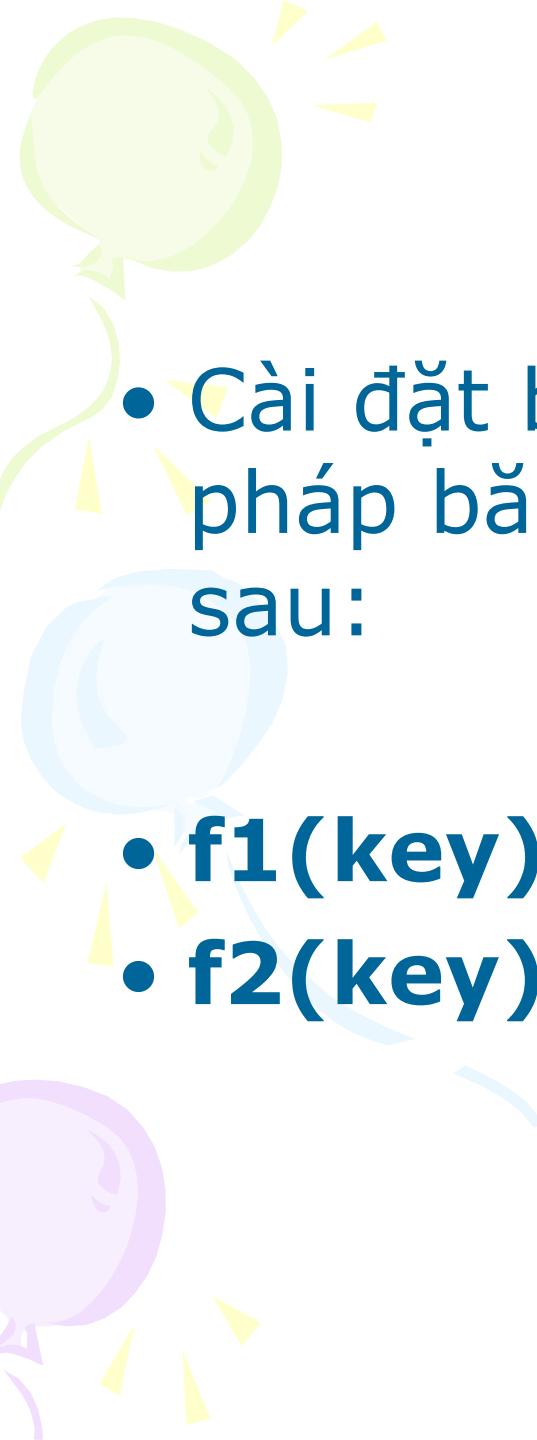
Băm 2 lần

- Sử dụng hàm băm thứ 2 $h_2(k)$ và xử lý xung đột bằng cách đặt item vào ô trống khả dụng đầu tiên trong chuỗi $(i + h_2(k)) \bmod N$
- Hàm $h_2(k)$ không thể trả về giá trị 0
- Bảng với k/thước N phải là số nguyên tố để cho phép thăm dò tất cả các ô

- Lựa chọn phổ biến của ánh xạ nén dữ liệu cho hàm băm thứ hai:

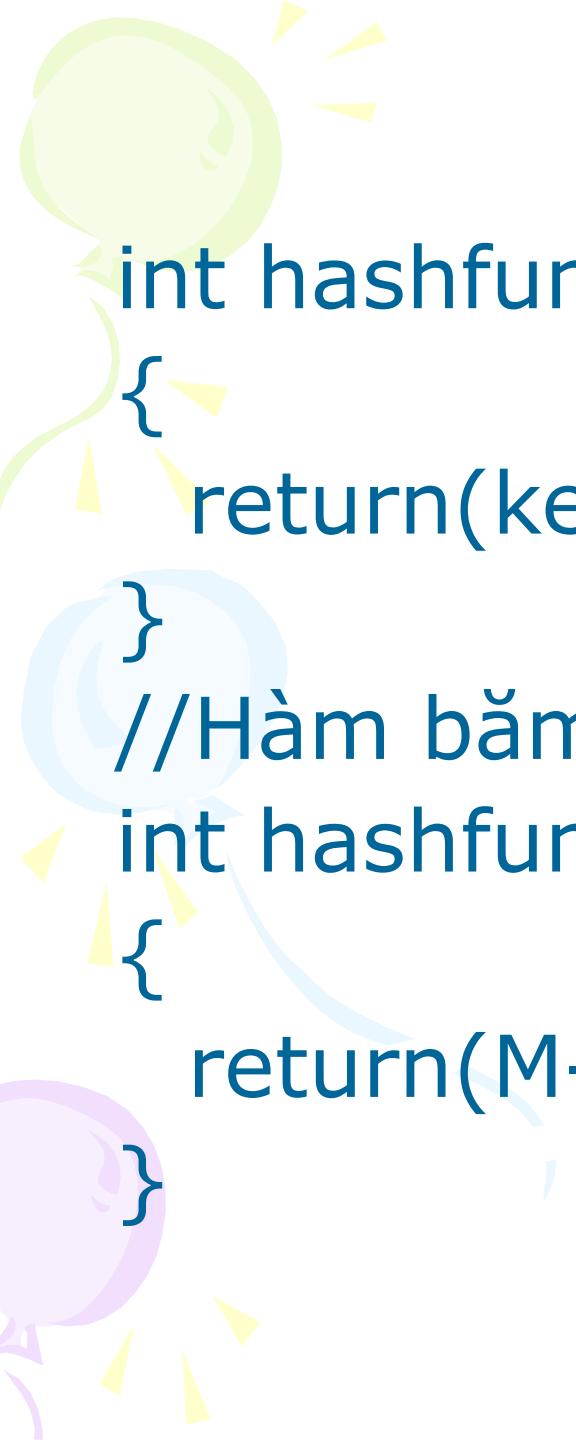
$$h_2(k) = q - k \bmod q$$

- Với
 - $q < N$
 - q là số nguyên tố



Exercise 14.5

- Cài đặt bảng băm ADT với phương pháp băm lại, sử dụng 2 hàm băm sau:
 - $f1(key) = key \% M$
 - $f2(key) = (M-2)-key \% (M-2)$



Hàm băm

```
int hashfunc(int key)
```

```
{
```

```
    return(key%M);
```

```
}
```

```
//Hàm băm thứ 2
```

```
int hashfunc2(int key)
```

```
{
```

```
    return(M-2 - key%(M-2));
```

```
}
```

Search

```
int search(int k) {  
    int i, j ;  
    i = hashfunc (k);  
    j = hashfunc2 (k);  
    while(hashtable[i].key!=k &&  
hashtable[i].key !=NULLKEY) {  
    //Bấm lại  
        i = (i+j) % M ;  
    }  
    if(hashtable[i].key==k) // found  
        return i;  
    else // not found  
    return M;  
}
```

Insert

```
int insert(int k) {  
    int i, j;  
    if(full()) {  
        printf("\n Hash table is full. Can not insert the  
        key %d ",k);  
        return M;  
    }  
    if (search (k) < M) {  
        printf ("This key exist in the hash table") ;  
        return M ;  
    }  
    i = hashfunc(k); j = hashfunc2(k) ;  
    while(hashtable[i].key !=NULLKEY) {  
        //Rehashing  
        i = (i+j) % M;  
    }  
    hashtable[i].key=k;  
    N=N+1;  
    return i;  
}
```