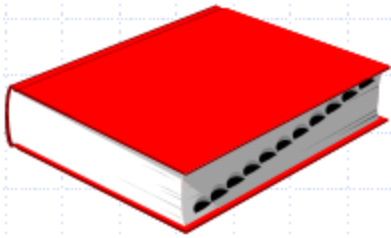
C Programming Basic – week 14

Nội dung

- Cấu trúc dữ liêu từ điển
- Bảng băm
- Hàm băm
- Bản đồ nén
- Xử lý xung đột
- Bài tập

Từ điển



- Từ điển mô hình hóa một tập hợp tìm kiếm được của các cặp khóa - phần tử
- Các thao tác chính của từ điển bao gồm tìm kiếm, chèn, và xóa phần tử
- Khóa là duy nhất nhưng các phần tử có thể có cùng khóa
- Ứng dụng:
  - Danh bạ địa chỉ
  - Xác thực thẻ tín dụng
  - Ánh xạ tên miền (vd csci260.net) với địa chỉ IP (vd 128.148.34.101)

Các phương thức

- findElement(k): nếu từ điển chứa phần tử với khóa k, trả về phần tử, nếu không trả về phần tử đặc biệt NO_SUCH_KEY
- insertItem(k, o): thêm cặp (k, o) vào từ điển
- removeElement(k): nếu từ điển chứa phần tử với khóa k, xóa khỏi từ điển và trả về phần tử, nếu không trả về phần tử đặc biệt NO_SUCH_KEY
- size(), isEmpty()
- keys(), elements()

Key-Indexed Dictionaries

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

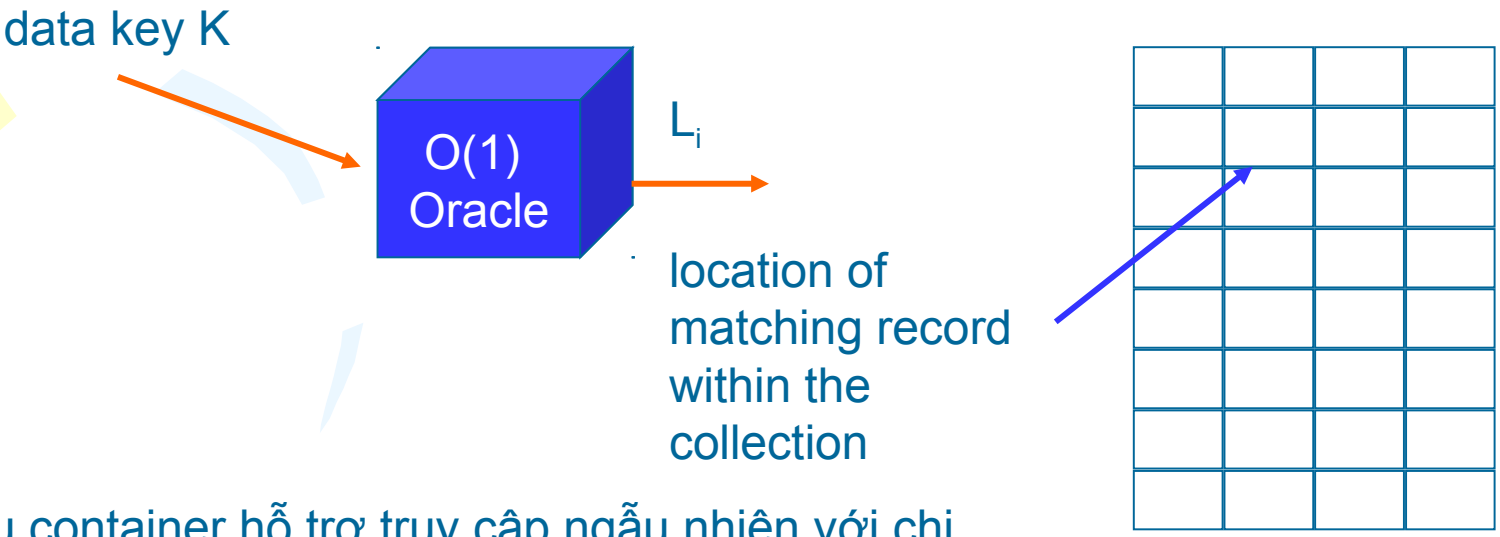
A[]

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

Space-efficient only if the cardinality of the set is close to N

Tìm kiếm không so sánh

- Làm thế nào để tìm kiếm mà không thực hiện so sánh các phần tử?
- Nếu chúng ta có một "oracle" nhận vào khóa của một giá trị dữ liệu và tính toán, trong thời gian tối đa cho trước, vị trí mà khóa đó xuất hiện trong tập dữ liệu?



Nếu container hỗ trợ truy cập ngẫu nhiên với chi phí Θ(1), tương tự mảng, ta có chi phí tìm kiếm Θ(1).

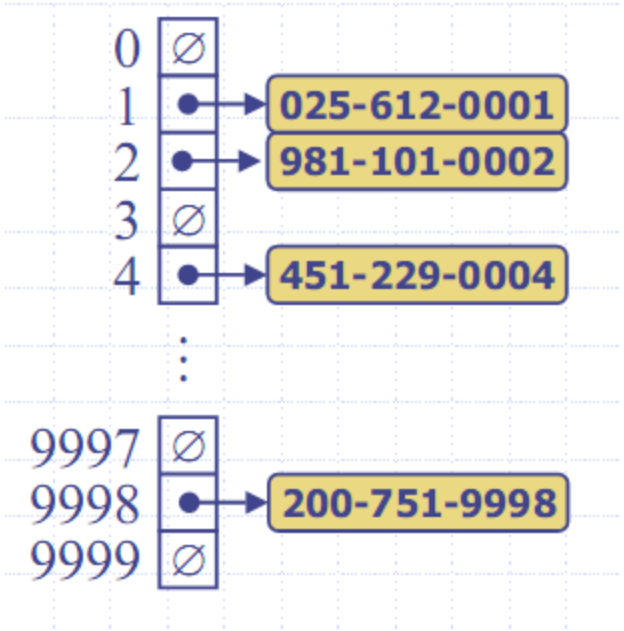
Hàm băm và bảng băm

- Một cách hiệu quả để cài đặt từ điển là bảng băm
- Sử dụng một bảng (hoặc danh sách) kích thước N (bảng)
  - Cần chia các khóa vào khoảng [0,N-1]
  - Xung đột xảy ra khi các phần tử có cùng khóa
- Khóa không phải khi nào cũng là số nguyên hoặc ở trong khoảng [0,N-1]
- Một bảng băm cho một loại khóa bao gồm
  - Hàm băm h
  - Mảng (bảng) kích thước N
- Khi cài đặt từ điển với bảng băm, mục đích là lưu trữ cặp (k, o) tại chỉ mục $i = h(k)$

VD

- Khi thiết kế bảng băm lưu trữ các đối tượng (SIN, Name), với SIN (social insurance number) là số nguyên dương 9 chữ số
- Bảng băm sử dụng một mảng kích cỡ N = 10,000 và hàm băm

h(x) = 4 chữ số cuối của x



Hàm băm

- Hàm băm h ánh xạ khóa có kiểu nhất định thành các số nguyên trong khoảng [0, N – 1]
- VD:

  h(x) = x mod N là hàm băm cho các khóa số nguyên

  Số nguyên h(x) được gọi là giá trị băm của khóa x
- Hàm băm thường bao gồm hai hàm thành phần
  - Bản đồ mã băm

  h1:keys → integers
  - Bản đồ nén

  h2: integers → [0, N – 1]

Bản đồ mã băm

- Ép kiểu số nguyên
  - Các bit của khóa được coi như các số nguyên
  - Phù hợp cho khóa có chiều dài ngắn hơn số bit của kiểu số nguyên
  - VD:
    - 'A' -> 65
    - 'N' ->78
- Tổng thành phần
  - Phân vùng các bit của khóa thành các thành phần có chiều dài xác định (vd 16 hoặc 32 bit) và tính tổng các thành phần
  - Phù hợp với khóa số có chiều dài xác định và ≥ số bit của kiểu số nguyên

$$x = \underbrace{(x_1}_{32\text{ bits}}, \underbrace{x_2}_{32\text{ bits}}, \ldots, \underbrace{x_{n-1})}_{32\text{ bits}} \Rightarrow h_1(x) = \sum_{i=0}^{n-1} x_i$$

Bản đồ mã băm (2)

- Tích lũy đa thức
  - Phân chia các bit của một khóa vào chuỗi các thành phần có độ dài xác định (vd 8, 16 hoặc 32 bit)

    $a_0\ a_1\ \ldots\ a_{n-1}$
  - Tính đa thức

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

    tại giá trị z xác định, bỏ qua tràn số
  - Đặc biệt phù hợp với chuỗi (vd chọn $z = 33$ sinh ra tối đa 6 xung đột trong tập hợp 50,000 từ tiếng Anh)

Exercise 14.1

- Viết ba hàm cài đặt ba phương pháp bản đồ mã băm
- Khóa đầu vào của ép kiểu số nguyên và tích lũy đa thức là chuỗi
- Khóa đầu vào của tổng thành phần là kiểu số nguyên lớn *long*

Bản đồ nén

- Kết quả của bản đồ mã băm cần phải chuẩn hóa về [0,N-1]
- Phương pháp chia:
  - $h_2(y) = |y| \bmod N$
  - Kích thước N của bảng băm thường được chọn là số nguyên tố
- Nhân, cộng, và chia (MAD):
  - $h2(y) = |ay + b| \bmod N$
  - a và b là các số nguyên không âm sao cho $a \bmod N \neq 0$
  - Ngược lại, mọi số nguyên được ánh xạ thành b

Cài đặt bảng băm

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

Phương pháp chia

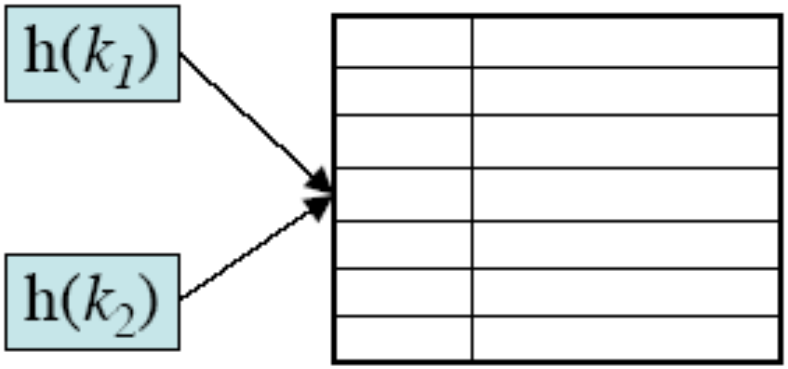
```
void init_table(element ht[])
{
   int i;
   for (i=0; i<TABLE_SIZE; i++)
     ht[i].key[0]=NULL;
}

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

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

Phân giải xung đột

- Xung đột xuất hiện khi $k_1 \neq k_2$ nhưng $h(k_1) = h(k_2)$
- Kết quả là hàm *insertItem*() và *findElement*() trở nên phức tạp hơn
- Chiến thuật phân giải xung đột
  - Địa chỉ đóng (bảng băm mở) - các slot ngoài $h(k)$ bị đóng và không sử dụng được
  - Địa chỉ mở (bảng băm đóng)- tìm một vị trí trống khác trong bảng



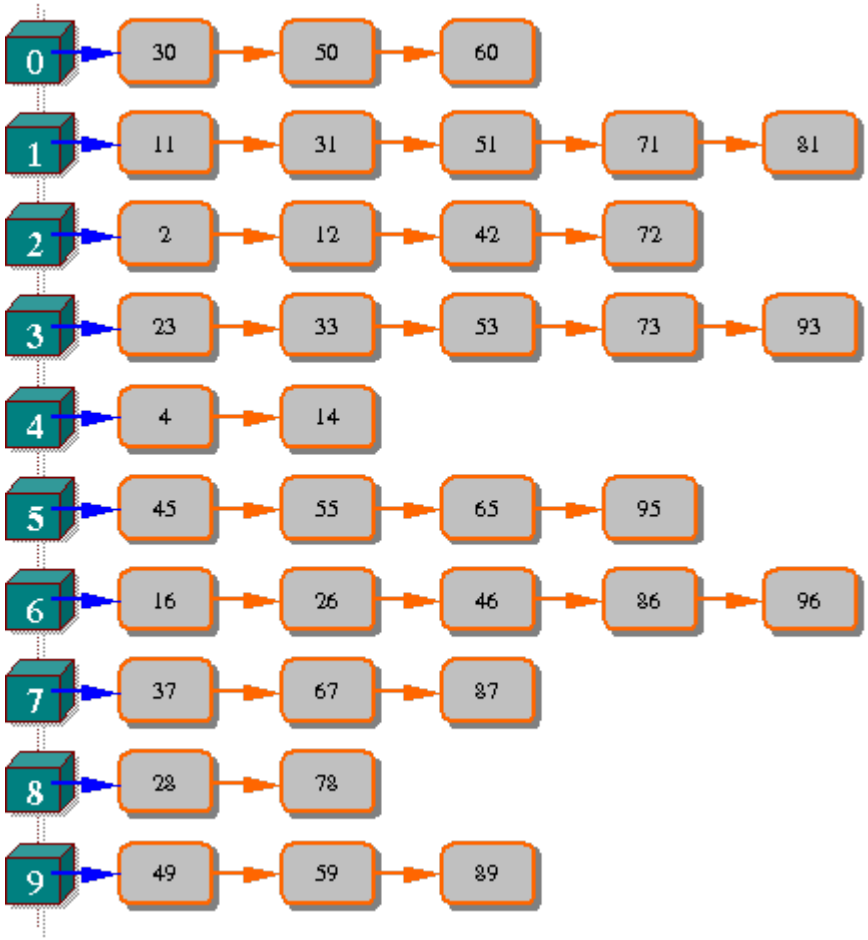
Cấu trúc dữ liệu bảng băm

- Bảng băm mở:
  - Phương pháp chuỗi xích
- Bảng băm đóng
  - Mở tuyến tính
  - Mở bậc hai
  - Băm kép

Cấu trúc chuỗi xích

- Mảng các con trỏ
- Mỗi con trỏ quản lý một danh sách liên kết tương ứng với một bucket (địa chỉ)
- VD về bảng băm chuỗi xích với hàm băm

  N mod 10



Exercise 14.2

- Cài đặt cấu trúc bảng băm chuỗi xích với các thao tác sau:
  - Init
  - Hash function
  - Insert (cho khóa và phần tử)
  - Search, Delete (khóa cho trước)
  - IsEmpty
  - Clear
  - Traverse

Khai báo

Data structure declaration

```
#define B ...  // size of hash table
typedef ... KeyType;  // int
typedef struct Node
{
   KeyType Key;
   // Add new fields if it is necessary
   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 kiếm

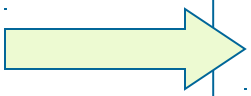
```
int Search(KeyType X)  {
   Position P;
   int Found=0;
   //Go to bucket at H(X)
   P=D[H(X)];
   //Traverse through the list at bucket
   H(X)
   while((P!=NULL) && (!Found))
      if (P->Key==X) Found=1;
    else P=P->Next;
  return Found;
}
```

Chèn phần tử

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

Xóa phần tử

```
void DeleteSet(ElementType X){
    int Bucket, Done;
    Position P,Q;
    Bucket=H(X);
    // If list has already existed
    if (D[Bucket]!=NULL) {
    // if X at the head of the list
    if D[Bucket]->Key==X)
    {
        Q=D[Bucket];
        D[Bucket]=D[Bucket]-
    >Next;
        free(Q);
  }
```



```
else { // Search for X
    Done=0;
    P=D[Bucket];
    while ((P->Next!=NULL) && (!
    Done))
        if (P->Next->Key==X)
            Done=1;
        else P=P->Next;
    if (Done) { // If found
        // Delete P->Next
        Q=P->Next;
        P->Next=Q->Next;
        free(Q);
    }
}
}
}
```

Kiểm tra rỗng

Verify if a bucket is empty

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

Verify if the table is empty

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

Xóa 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)
   {
       // Assume that the key is of int type
       printf("%3d", p->key);
       p= p->next;
   }
}
```

Duyệt bảng băm

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

Exercise 14.3

- Xây dựng danh bạ điện thoại
- Khai báo cấu trúc chứa ít nhất “name,” “telephone number,” và “e-mail address”, và lưu trữ tối đa 100 phần tử.
- (1) Đọc 10 phần tử từ tệp, lưu trữ trong bảng băm có “e-mail address” là khóa.
- (2) Định nghĩa một hàm băm phù hợp

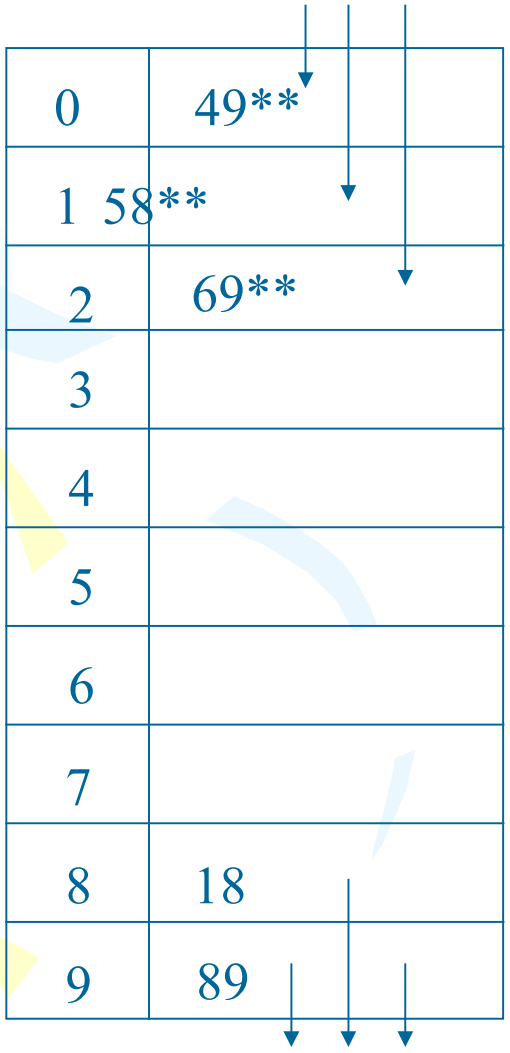
Mở tuyến tính

- Tính f(x)
- Kiểm tra bucket

  ht[(f(x)+j)%TABLE_SIZE]

  $0 \le j \le$ TABLE_SIZE
  - Bucket chứa x.
  - Bucket chứa xâu rỗng
  - Bucket chứa xâu khác x
  - Return ht[f(x)]

Mở tuyến tính - VD



Mở tuyến tính $f(i) = i$.

Bảng băm kích thước $T = 10$, các mục 89, 18, 49, 58, và 69 đã được chèn
Hàm băm $h(key) = key\%10$.

Chú ý: Trong thực tế, kích thước bảng băm N thường là số nguyên tố

Exercise 14.4

- Cài đặt bảng băm với phương pháp mở tuyến tính

Khai báo

```
#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/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

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

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

Tìm kiếm

```
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;
}
```

Chèn

```
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;
}
```

Xóa một khóa

```
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);
}
```

Mở bậc hai

- Mở tuyến tính có xu hướng phân cụm
  - Tìm kiếm chậm
- Loại bỏ vấn đề phân cụm
- dùng hàm xung đột bậc hai, $f(i) = i^2$
- không đảm bảo tìm thấy chỗ trống nếu bảng đầy hơn một nửa trừ khi kích thước bảng là số nguyên tố

Exercise 14.5

- Cài đặt bảng băm với phương pháp mở 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 kép

- Băm kép sử dụng một hàm băm thứ hai $h_2(k)$ và xử lý xung đột bằng cách đặt một phần tử ở vị trí còn trống đầu tiên trong chuỗi

  $(i + h_2(k)) \bmod N$
- Hàm băm thứ hai $\boldsymbol{h_2(k)}$ luôn có giá trị khác 0
- Kích thước bảng ***N*** phải là số nguyên tố để mở tất cả các ô

- Bản đồ nén của hàm băm thứ hai

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

- với
  - q < N
  - q là SNT

Exercise 14.6

- Cài đặt bảng băm với phương pháp băm lại, sử dụng hai hàm băm:

- **f1(key)= key % M**
- **f2(key) =(M-2)-key %(M-2)**

Hash functions

```
int hashfunc(int key)
{
  return(key%M);
}
//Secondary function
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){
   //Rehashing
      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;
}
```