|  |
| --- |
| Đại học Quốc gia TP.HCM  **BỘ GIÁO DỤC VÀ ĐÀO TẠO**  **TRƯỜNG ĐẠI HỌC AN GIANG**  **🙡🕮🙣**    **BÁO CÁO TIỂU LUẬN**  **MÔN PHÂN TÍCH VÀ THIẾT KẾ GIẢI THUẬT**  **Lớp:** **DH20TH1**  **GIẢNG VIÊN: Trương Thị Diễm**  NHÓM 8: |

**Bảng Băm(Hash table)**

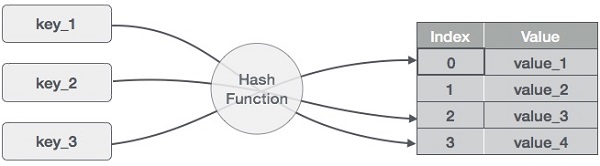
**Giới Thiệu**

Bảng băm là một cấu trúc dữ liệu lưu trữ dữ liệu theo cách liên kết. Trong bảng băm, dữ liệu được lưu trữ ở định dạng mảng, trong đó mỗi giá trị dữ liệu có giá trị chỉ mục duy nhất của riêng nó. Việc truy cập dữ liệu trở nên rất nhanh nếu chúng ta biết chỉ mục của dữ liệu mong muốn.

Do đó, nó trở thành một cấu trúc dữ liệu trong đó các thao tác chèn và tìm kiếm rất nhanh bất kể kích thước của dữ liệu. Bảng băm sử dụng một mảng làm phương tiện lưu trữ và sử dụng kỹ thuật băm để tạo chỉ mục nơi một phần tử sẽ được chèn vào hoặc được định vị từ đó.

**1.Băm**

Hashing là một kỹ thuật để chuyển đổi một loạt các giá trị khóa thành một loạt các chỉ mục của một mảng. Chúng ta sẽ sử dụng toán tử modulo để nhận một loạt các giá trị chính. Hãy xem xét một ví dụ về bảng băm có kích thước 20 và các mục sau đây sẽ được lưu trữ. Mục ở định dạng (khóa, giá trị).

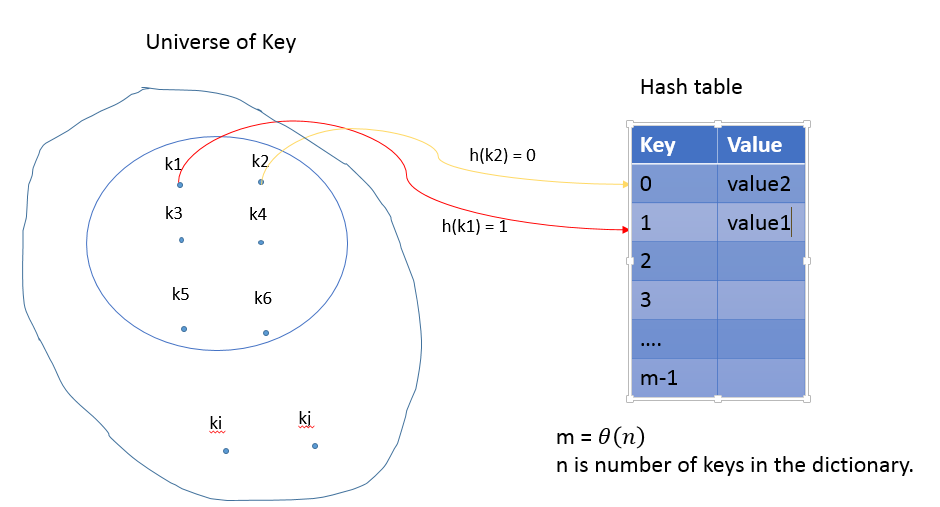


* (1,20)
* (2,70)
* (42,80)
* (4,25)
* (12,44)
* (14,32)
* (17,11)
* (13,78)
* (37,98)

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.No.** | **Chìa khóa** | **Băm** | **Chỉ mục mảng** |
| 1 | 1 | 1% 20 = 1 | 1 | |
| 2 | 2 | 2% 20 = 2 | 2 | |
| 3 | 42 | 42% 20 = 2 | 2 | |
| 4 | 4 | 4% 20 = 4 | 4 | |
| 5 | 12 | 12% 20 = 12 | 12 | |
| 6 | 14 | 14% 20 = 14 | 14 | |
| 7 | 17 | 17% 20 = 17 | 17 | |
| 8 | 13 | 13% 20 = 13 | 13 | |
| 9 | 37 | 37% 20 = 17 | 17 | |

**2.Hash function**

(Động từ "hash" có nghĩa là cắt nhỏ ra thành nhiều mảnh, sau đó mix lại với nhau. Đây là 1 từ có nguồn gốc từ tiếng Pháp cổ, "hachet")

* [](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/2rxgwjpe9c_hash.PNG)Mục đích của 1 hàm hash function *h* này là giảm không gian key (integer) xuống một kích thước hợp lý (ta gọi là m), m cũng sẽ là kích thước của table mà chúng ta sử dụng. Đồng nghĩa với việc chúng ta sẽ làm giảm kích thước của "direct access table" đã nhắc tới ở trên. Các bạn có thể xem hình dưới đây để hiểu hơn cách làm của hash function.

Ở đây, ta insert 2 cặp item (k1, value1) và (k2, value2) vào bảng. Hàm h(k1) tính ra vị trí của key k1 là 1, nên value1 sẽ nằm tương ứng với index 1. Hàm h(k2) = 0, nên value2 sẽ nằm tương ứng với index bằng 0.

* Với m xấp xỉ với n. Có thể là m = n/2 hay n/3, v.v... Chúng ta sẽ tính m ở các phần sau.
* Như vậy chúng ta thấy có thể xảy ra tình huống với hai key khác nhau ki != kj, nhưng hàm hash function có thể trả về cùng một kết quá, tức là h(ki) = h(kj) , khi đó hai item tuy có giá trị key khác nhau, nhưng sẽ cùng nằm chung một nơi, do kết quả của hàm hash function là như nhau. Tình huống này gọi là **collision**. Để giải quyết vấn đề này chúng ta có 2 cách.

### ***a.Một hàm băm tốt phải thỏa mãn các điều kiện sau:***

Tính toán nhanh.   
Các khoá được phân bố đều trong bảng.  
Ít xảy ra đụng độ.  
Xử lý được các loại khóa có kiểu dữ liệu khác nhau.

### ***b.Ứng dụng của Hàm băm***

Các hàm băm được ứng dụng trong nhiều lĩnh vực, chúng thường được thiết kế phù hợp với từng ứng dụng. Ví dụ, các hàm băm mật mã học giả thiết sự tồn tại của một đối phương – người có thể cố tình tìm các dữ liệu vào với cùng một giá trị băm. Một hàm băm tốt là một phép biến đổi “một chiều”, nghĩa là không có một phương pháp thực tiễn để tính toán được dữ liệu vào nào đó tương ứng với giá trị băm mong muốn, khi đó việc giả mạo sẽ rất khó khăn. Một hàm một chiều mật mã học điển hình không có tính chất hàm đơn ánh và tạo nên một hàm băm hiệu quả; một hàm trapdoor mật mã học điển hình là hàm đơn ánh và tạo nên một hàm ngẫu nhiên hiệu quả.

Bảng băm, một ứng dụng quan trọng của các hàm băm, cho phép tra cứu nhanh một bản ghi dữ liệu nếu cho trước khóa của bản ghi đó (Lưu ý: các khóa này thường không bí mật như trong mật mã học, nhưng cả hai đều được dùng để “mở khóa” hoặc để truy nhập thông tin.) Ví dụ, các khóa trong một từ điển điện tử Anh-Anh có thể là các từ tiếng Anh, các bản ghi tương ứng với chúng chứa các định nghĩa. Trong trường hợp này, hàm băm phải ánh xạ các xâu chữ cái tới các chỉ mục của mảng nội bộ của bảng băm.

Các hàm băm dành cho việc phát hiện và sửa lỗi tập trung phân biệt các trường hợp mà dữ liệu đã bị làm nhiễu bởi các quá trình ngẫu nhiên. Khi các hàm băm được dùng cho các giá trị tổng kiểm, giá trị băm tương đối nhỏ có thể được dùng để kiểm chứng rằng một file dữ liệu có kích thước tùy ý chưa bị sửa đổi. Hàm băm được dùng để phát hiện lỗi truyền dữ liệu. Tại nơi gửi, hàm băm được tính cho dữ liệu được gửi, giá trị băm này được gửi cùng dữ liệu. Tại đầu nhận, hàm băm lại được tính lần nữa, nếu các giá trị băm không trùng nhau thì lỗi đã xảy ra ở đâu đó trong quá trình truyền. Việc này được gọi là kiểm tra dư (redundancy check).

Các hàm băm còn được ứng dụng trong việc nhận dạng âm thanh, chẳng hạn như xác định xem một file MP3 có khớp với một file trong danh sách một loại các file khác hay không.

Thuật toán tìm kiếm xâu Rabin-Karp là một thuật toán tìm kiếm xâu ký tự tương đối nhanh, với thời gian chạy trung bình O(n). Thuật toán này dựa trên việc sử dụng băm để so sánh xâu.

**3.Đo tuyến tính**

Như chúng ta thấy, có thể xảy ra trường hợp kỹ thuật băm được sử dụng để tạo chỉ mục đã được sử dụng của mảng. Trong trường hợp như vậy, chúng ta có thể tìm kiếm vị trí trống tiếp theo trong mảng bằng cách nhìn vào ô tiếp theo cho đến khi tìm thấy ô trống. Kỹ thuật này được gọi là thăm dò tuyến tính.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.No.** | **Chìa khóa** | **Băm** | **Chỉ mục mảng** | **Sau khi dò tuyến tính, chỉ mục mảng** |
| 1 | 1 | 1% 20 = 1 | 1 | 1 |
| 2 | 2 | 2% 20 = 2 | 2 | 2 |
| 3 | 42 | 42% 20 = 2 | 2 | 3 |
| 4 | 4 | 4% 20 = 4 | 4 | 4 |
| 5 | 12 | 12% 20 = 12 | 12 | 12 |
| 6 | 14 | 14% 20 = 14 | 14 | 14 |
| 7 | 17 | 17% 20 = 17 | 17 | 17 |
| 8 | 13 | 13% 20 = 13 | 13 | 13 |
| 9 | 37 | 37% 20 = 17 | 17 | 18 |

**4.Hoạt động cơ bản**

Sau đây là các hoạt động chính cơ bản của bảng băm.

* **Tìm kiếm** - Tìm kiếm một phần tử trong bảng băm.
* **Chèn** - chèn một phần tử trong bảng băm.
* **xóa** - Xóa một phần tử khỏi bảng băm.
  1. **Mục dữ liệu**

Xác định một mục dữ liệu có một số dữ liệu và khóa, dựa vào đó việc tìm kiếm sẽ được tiến hành trong bảng băm.

struct DataItem {

int data;

int key;

};

* 1. **Phương pháp băm**

Xác định phương pháp băm để tính mã băm của khóa của mục dữ liệu.

int hashCode(int key){

return key % SIZE;

}

* 1. **Hoạt động tìm kiếm**

Bất cứ khi nào một phần tử được tìm kiếm, hãy tính toán mã băm của khóa được truyền và xác định vị trí phần tử bằng cách sử dụng mã băm đó làm chỉ mục trong mảng. Sử dụng thăm dò tuyến tính để đưa phần tử đi trước nếu phần tử không được tìm thấy trong mã băm đã tính.

Thí dụ

struct DataItem \*search(int key) {

//get the hash

int hashIndex = hashCode(key);

//move in array until an empty

while(hashArray[hashIndex] != NULL) {

if(hashArray[hashIndex]->key == key)

return hashArray[hashIndex];

//go to next cell

++hashIndex;

//wrap around the table

hashIndex %= SIZE;

}

return NULL;

}

* 1. **Chèn hoạt động**

Bất cứ khi nào một phần tử được chèn vào, hãy tính toán mã băm của khóa được truyền và xác định vị trí chỉ mục bằng cách sử dụng mã băm đó làm chỉ mục trong mảng. Sử dụng thăm dò tuyến tính cho vị trí trống, nếu một phần tử được tìm thấy trong mã băm được tính toán.

Thí dụ

void insert(int key,int data) {

struct DataItem \*item = (struct DataItem\*) malloc(sizeof(struct DataItem));

item->data = data;

item->key = key;

//get the hash

int hashIndex = hashCode(key);

//move in array until an empty or deleted cell

while(hashArray[hashIndex] != NULL && hashArray[hashIndex]->key != -1) {

//go to next cell

++hashIndex;

//wrap around the table

hashIndex %= SIZE;

}

hashArray[hashIndex] = item;

}

* 1. **Xóa hoạt động**

Bất cứ khi nào một phần tử bị xóa, hãy tính mã băm của khóa được truyền và xác định vị trí chỉ mục bằng cách sử dụng mã băm đó làm chỉ mục trong mảng. Sử dụng thăm dò tuyến tính để đưa phần tử đi trước nếu phần tử không được tìm thấy trong mã băm đã tính. Khi tìm thấy, hãy lưu trữ một mục giả ở đó để giữ nguyên hiệu suất của bảng băm.

Thí dụ

struct DataItem\* delete(struct DataItem\* item) {

int key = item->key;

//get the hash

int hashIndex = hashCode(key);

//move in array until an empty

while(hashArray[hashIndex] !=NULL) {

if(hashArray[hashIndex]->key == key) {

struct DataItem\* temp = hashArray[hashIndex];

//assign a dummy item at deleted position

hashArray[hashIndex] = dummyItem;

return temp;

}

//go to next cell

++hashIndex;

//wrap around the table

hashIndex %= SIZE;

}

return NULL;

}

**5. Bảng Băm Kết Nối 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 kết nối 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.

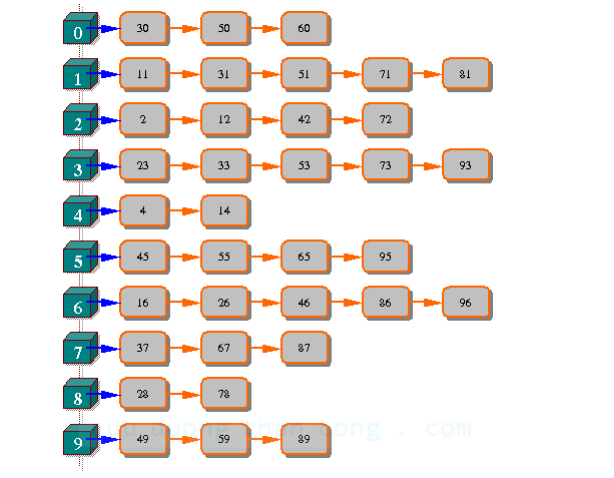
Để minh họa ta xét bảng băm có cấu trúc như sau:

- Tập khóa K: tập số tự nhiên

- Tập địa chỉ M: gồm 10 địa chỉ (M={0, 1, …, 9}

- Hàm băm h(key) = key % 10.

30, 50,60,11,21,31,…



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

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.

· Khi khởi động bảng băm, con trỏ đầu của các bucket là NULL.

Theo cấu trúc này, với tác vụ insert, hàm băm h(k) sẽ được dùng để tính địa chỉ của khoá k, tức là xác định 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

+ i=h(k) => thuoc danh sach thu I (bucket[i]

+ tim kiem khoa K tren danh sach bucket[i]

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

1. ***Khai báo cấu trúc bảng băm:***

#define M 100

struct nodes { int key; struct nodes \*next };

typedef struct nodes \*NODEPTR; //khai bao kieu con tro chi nut

/\*khai bao mang bucket chua M con tro dau cua Mbucket \*/

NODEPTR bucket[M];

BT: xay dung bang bam theo PP ket noi truc tiep

***b.Các phép toán:***

- Tính giá trị hàm băm: Giả sử chúng ta chọn hàm băm dạng %: h(key)=key % M.

- Phép toán initbuckets: khởi tạo các bucket băng Null.

- Phép toán emmptybucket(b): kiểm tra bucket b có bị rỗng không?

- Phép toán emmpty: Kiểm tra bảng băm có rỗng không?

- Phép toán insert: Thêm phần tử có khóa k vào bảng băm.

+ i=h(k)

+ ktra bucket [i]: neu rong =>cc o nho cho bucket, gan khoa k

them phan tu co khoa k vao ds theo thu tu tang dan.

- Phép toán remove: Xóa phần tử có khóa k trong bảng băm.

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

- Phép toán traversebucket: Xử lý tất cả các phần tử trong bucket b.

- Phép toán traverse: Xử lý tất cả các phần tử trong bảng băm.

- Phép toán search:Tìm kiếm một phần tử trong bảng băm, nếu không tìm thấy hàm này trả về hàm NULL, nếu tìm thấy hàm này trả về địa chỉ của phần tử có khóa k.

B1: Tìm danh sách liên kết có thể chứa khóa k b = h(k); p = bucket[b]; B2: Tìm khóa k trong danh sách liên kết p.

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

**//+// Khai báo cấu trúc bảng băm**

typedef struct tagNODE  
{  
        int               key;  
        tagNODE \*next  
}NODE, \*NODEPTR;  
   
NODEPTR bucket[M];           /\* khai bao mang bucket chua M con tro dau cua Mbucket \*/

**//+// 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( )  
{  
        for (int b=0;  b<M;  b++)  
                  ;  
        bucket[b] = NULL;  
}

**//+// Phép toán kiểm tra EmptyBucket:**Kiểm tra bucket b có bị rỗng không?

int IsEmptyBucket (int b)  
{  
        return (bucket[b]==NULL? TRUE : FALSE);  
}

**//+// Phép toán kiểm tra Empty:**Kiểm tra bảng băm có rỗng không?

int IsEmpty( )  
{  
      for (int 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 = 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 = hashfunc(k);  
        NODEPTR p = hashbucket(k);  
        NODEPTR 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)  
{  
        //q la nut truoc,p la nut sau  
        NODEPTR q = NULL;  
        NODEPTR 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( )  
{  
        for (int b = 0; b<M ; b++)  
                ClearBucket(b);  
}

**//+// Phép toán TraverseBucket:** Duyệt các phần tử trong bucket b.

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

**//+// Phép toán Traverse:**Duyệt toàn bộ bảng băm.

void Traverse( )  
{  
        for (int b = 0;n<M; b++){  
                printf("\nButket %d:",b);  
                TraverseBucket(b);  
        }  
}

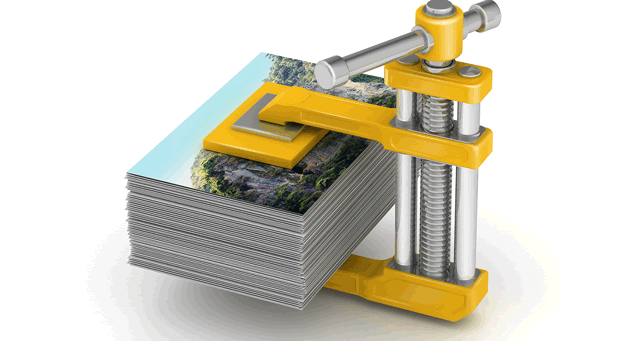
**//+// Phép toán Search:**Tìm kiếm một phần tử trong bảng băm,nếu không tìm thấy hàm này trả về hàm NULL,nếu tìm thấy hàm này trả về con trả chỉ tìm phần tử tìm thấy.

NODEPTR Search(int k)  
{  
        int b = hashfunc (k);  
        NODEPTR 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  
                return p;  
}

**6.Ưu Điểm và Nhược Điểm của Bảng Băm**

a.Ưu Điểm

Hash được sử dụng cho mật mã vì nó che dấu dữ liệu gốc với một giá trị khác. Hàm băm có thể được dùng để tạo một giá trị mà bạn chỉ có thể giải mã bằng cách tra cứu từ bảng băm. Bảng này có thể là một mảng, cơ sở dữ liệu hoặc cấu trúc dữ liệu. Một hàm băm mật mã tốt là không thể đảo ngược.



[Các loại nén khác nhau](https://quantrimang.com/co-che-hoat-dong-cua-cong-cu-nen-tap-tin-130435), chẳng hạn như nén phương tiện và nén ảnh bị mất dữ liệu (lossy) có thể kết hợp hàm băm để giảm kích thước file. Bằng cách hash dữ liệu thành giá trị nhỏ hơn, file phương tiện có thể được nén thành các phần nhỏ hơn. Kiểu hash một chiều này không thể đảo ngược nhưng nó có thể tạo dữ liệu gần bằng dữ liệu gốc nhưng yêu cầu ít dung lượng đĩa hơn

b.Nhược Điểm

Nếu phân bố không đều vào các khóa mục sẽ dẫn đến tình trạng lãng phí các khóa mục

**-HẾT RỒI HIHI-**