**HỘI THẢO KHOA HỌC**

**CHUYÊN ĐỀ**

## CẤU TRÚC DỮ LIỆU BĂM (HASHING)

**MỤC LỤC**

[A. PHẦN MỞ ĐẦU 3](#_Toc15829503)

[B. PHẦN NỘI DUNG 4](#_Toc15829511)

[I. CẤU TRÚC DỮ LIỆU BĂM (HASHING) 4](#_Toc15829512)

[1. Khái niệm về Hashing 4](#_Toc15829513)

[2. Hàm băm (Hash fuction) 6](#_Toc15829514)

[2.1. Khái niệm 6](#_Toc15829515)

[2.2. Xây dựng hàm băm 6](#_Toc15829519)

[2.3. Xây dựng hàm băm mới 8](#_Toc15829526)

[2.3.1. Hàm băm Robert Sedgwicks (RS hash function) 8](#_Toc15829527)

[2.3.2. Hàm băm Polynomimal (Hàm băm đa thức) 8](#_Toc15829531)

[2.3.3. Hàm băm Cyclic shift 9](#_Toc15829535)

[2.4. Các phương pháp giải quyết va chạm 10](#_Toc15829539)

[2.4.1. Separate chaining (phương pháp kết nối) 10](#_Toc15829540)

[2.4.2. Linear probing (dò tuyến tính) 11](#_Toc15829541)

[2.4.3. Quadratic Probing (dò bậc 2) 13](#_Toc15829545)

[2.4.4. Double hashing (Băm kép) 14](#_Toc15829549)

[II. BÀI TẬP ÁP DỤNG 16](#_Toc15829553)

[1. Bài 1: Zero Quantity Maximization 16](#_Toc15829554)

[1.1. Đề bài 16](#_Toc15829555)

[1.2. Hướng dẫn giải thuật. 17](#_Toc15829559)

[1.3. Chương trình. 17](#_Toc15829560)

[1.4. Test 18](#_Toc15829561)

[2. Bài 2: Xâu con 18](#_Toc15829562)

[2.1. Đề bài 18](#_Toc15829563)

[2.2. Hướng dẫn thuật toán: 18](#_Toc15829566)

[2.3. Cài đặt: 19](#_Toc15829569)

[2.4. Test 20](#_Toc15829570)

[3. Bài 3: Xâu con đối xứng dài nhất 20](#_Toc15829571)

[3.1. Đề bài 20](#_Toc15829572)

[3.2. Hướng dẫn giải thuật. 21](#_Toc15829575)

[3.3. Chương trình. 21](#_Toc15829576)

[3.4. Test 22](#_Toc15829577)

[4. Bài 4: Xâm nhập mật khẩu 23](#_Toc15829578)

[4.1. Đề bài 23](#_Toc15829579)

[4.2. Hướng dẫn giải thuật 24](#_Toc15829584)

[4.3. Chương trình. 24](#_Toc15829588)

[4.4. Test 25](#_Toc15829589)

[5. Bài 5: Chuỗi abc 25](#_Toc15829590)

[5.1. Đề bài 26](#_Toc15829591)

[5.2. Hướng dẫn giải thuật: 27](#_Toc15829595)

[5.3. Cài đặt 27](#_Toc15829598)

5.4. Test 27

[6. Một vài bài toán khác 28](#_Toc15829599)

[C. PHẦN KẾT LUẬN 29](#_Toc15829600)

[D. TÀI LIỆU THAM KHẢO 30](#_Toc15829601)

# phần MỞ ĐẦU

Trong Tin học, khi thiết kế chương trình cho một bài toán, việc chọn cấu trúc dữ liệu là vấn đề rất quan trọng vì mỗi loại cấu trúc dữ liệu phù hợp với một vài loại bài toán ứng dụng khác nhau. Một cấu trúc dữ liệu được chọn cẩn thận sẽ cho phép thực hiện [thuật toán](https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n) hiệu quả hơn.

Trong chương trình bồi dưỡng học sinh giỏi, vấn đề sử dụng cấu trúc dữ liệu đặc biệt để giải các bài là một trong những vấn đề rất hay nhưng cũng rất khó. Hiện nay, phổ biến rất nhiều loại cấu trúc dữ liệu khác nhau như:

1. Ngăn xếp (stack)
2. Hàng đợi (queue)
3. Băm (Hashing)
4. Đống (heap)
5. RMQ (Range Minimum Query)
6. IT (Interval Tree)
7. BIT (Binarry Indexed Tree)

…

Trong bài viết này, tôi chỉ trình bày cấu trúc dữ liệu Băm, không mang nặng vấn đề lí thuyết đầy đủ, chỉ ở mức tổng quát để đi đến các ứng dụng giải các bài toán cụ thể.

# PHẦN NỘI DUNG

## CẤU TRÚC DỮ LIỆU BĂM (HASHING)

### Khái niệm về Hashing

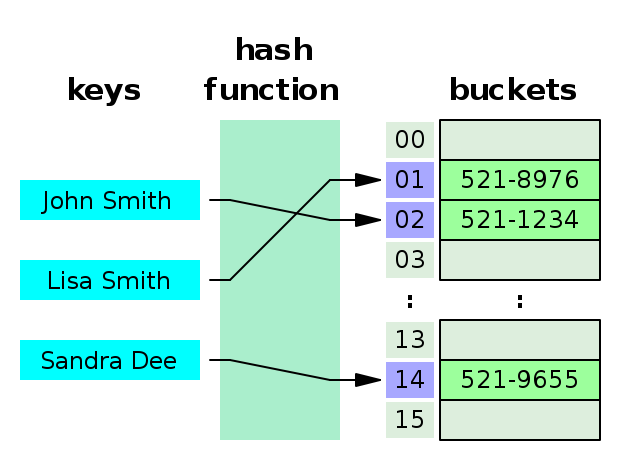
Hashing là một cấu trúc dữ liệu quan trọng được thiết kế để sử dụng một hàm đặc biệt gọi là hàm Hash, được sử dụng để ánh xạ một giá trị xác định với một khóa cụ thể để truy cập các phần tử nhanh hơn. The efficiency of mapping depends of the efficiency of the hash function used.Hiệu quả của ánh xạ phụ thuộc vào hiệu quả của hàm băm được sử dụng.

Let a hash function H(x) maps the valueMột số ví dụ về sử dụng cấu trúc dữ liệu băm trong thực tế:

- Trong trường đại học, mỗi sinh viên được chỉ định một mã sinh viên không giống nhau và qua mã sinh viên đó có thể truy xuất các thông tin của sinh viên đó.

- Trong thư viện, mỗi một cuốn sách có một mã số riêng và mã số đó có thể được dùng để xác định các thông tin của sách, chẳng hạn như vị trí chính xác của sách trong thư viện hay thể loại của sách đó,…

Trong cả 2 ví dụ trên, các sinh viên và các cuốn sách được “băm” thành các mã số duy nhất (không trùng lặp).



Giả sử rằng bạn có một đối tượng và bạn muốn gán cho nó một cái khóa (key) để giúp tìm kiếm dễ dàng hơn. Để lưu giữ cặp <khóa, giá trị> (nó thường được gọi là <key, value>), bạn có thể sử dụng mảng bình thường để làm việc này. Với chỉ số mảng là khóa và giá trị tại chỉ số đó là giá trị tương ứng của khóa. Tuy nhiên, trong trường hợp phạm vi của khóa lớn và không thể sử dụng chỉ số mảng được, khi đó bạn sẽ cần tới “băm” (hashing).

Trong hashing, các key có giá trị lớn sẽ được đưa về giá trị nhỏ hơn bằng cách sử dụng hàm băm(hash functions). Các giá trị sau đó được lưu trong một cấu trúc dữ liệu gọi là bảng băm(hash tables). Ý tưởng của hashing là đưa các cặp <key, value> về một mảng thống nhất. Mỗi phần tử sẽ được gán một khóa định danh (khóa có được sau khi dùng hàm băm). Bằng việc sử dụng khóa định danh đó, chúng ta có thể truy cập trực tiếp tới nó với độ phức tạp **O(1)**. Thuật toán băm sẽ sử dụng khóa băm để tính toán ra khóa định danh của các phần tử hoặc thêm vào bảng băm.

Việc hashing được thực hiện qua 2 bước:

Một phần tử sẽ được chuyển đổi thành 1 số nguyên bằng việc sử dụng hàm băm. Phần tử này được sử dụng như một chỉ mục để lưu trữ phần tử gốc và nó sẽ được đưa vào bảng băm.

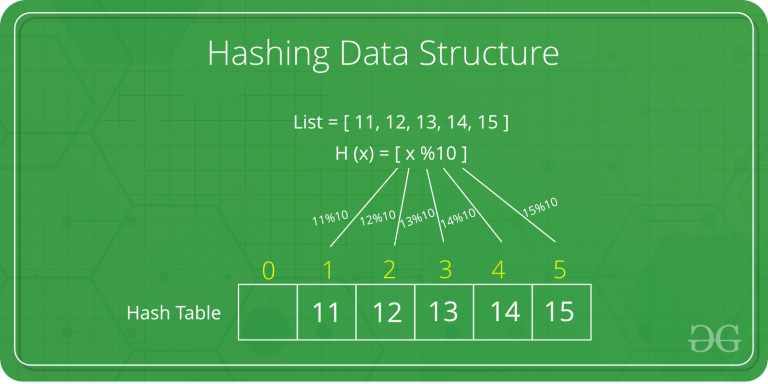
Phần tử sẽ được lưu giữ trong bảng băm, nó có thể được truy xuất nhanh bằng khóa băm:

*hash = hashfunc(key)*

*index = hash % array\_size*

Theo cách trên thì việc băm sẽ phụ thuộc vào kích thước mảng table\_size. Chỉ số index sau đó được đưa về [0; table\_size-1] bằng việc sử dụng toán tử chia lấy dư %.

**Ví dụ:** Cho danh sách các giá trị là [11,12,13,14,15] thì nó sẽ được lưu trữ tại các vị trí {1,2,3,4,5} trong mảng hoặc bảng Hash tương ứng với hàm băm H(x) ánh xạ giá trị xat the index **x%10** in an Array. tại chỉ số **x% 10** trong một mảng.

For example if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.

### Hàm băm (Hash fuction)

#### Khái niệm

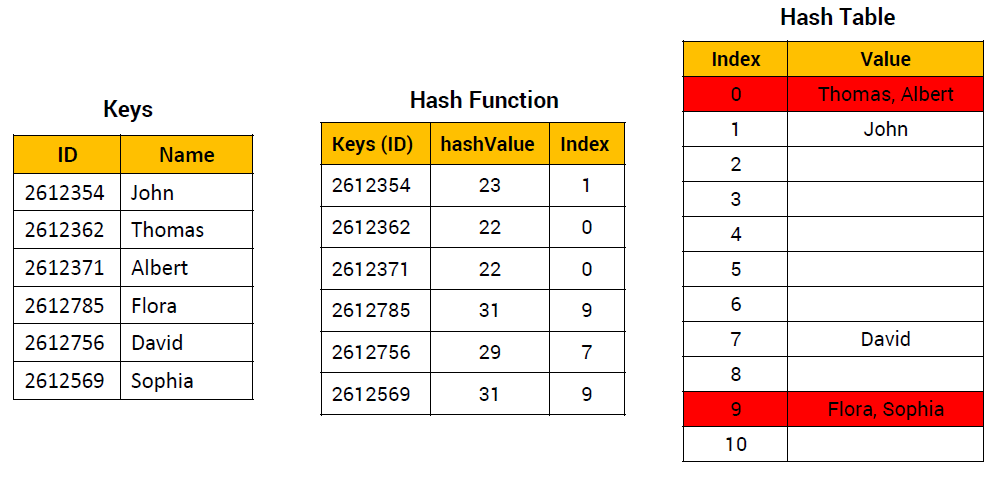
Hàm băm là hàm biến giá trị khóa (số, chuỗi, kí hiệu, …) thành địa chỉ/ chỉ mục trong bảng băm. Không có một hàm băm tối ưu cho mọi trường hợp.

Có nhiều cách xây dựng hàm băm. Một hàm băm tốt cần thỏa mãn các yêu cầu sau :

* Tính toán nhanh.
* Ít xảy ra đụng độ.
* Các khóa được phân bổ đều trong bảng băm.

#### Xây dựng hàm băm

**\*Cách 1 :** Cách đơn giản để xây dựng hàm băm là ứng với mỗi khóa (key), sau khi tính tổng các kí tự ta dùng phép mod cho một số N. Với N là phần tử trong mảng giá trị.



Một số lưu ý khi chọn số N  để hàm băm “Sạch”:

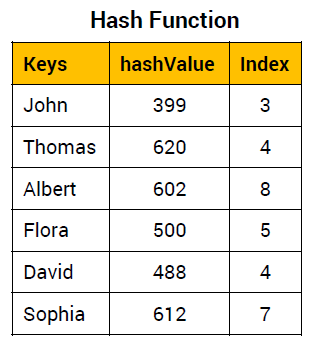
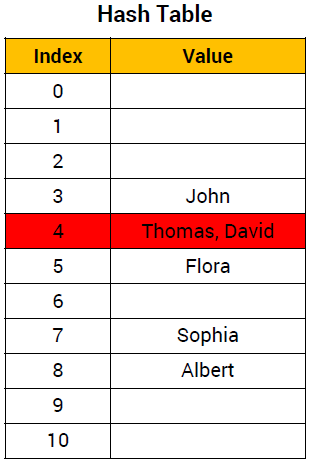
* + N không nên là số lũy thừa của 2.
  + N nên là số nguyên tố không gần với lũy thừa của 2.

**\*Cách 2 :** Một cách khác, thay vì băm ID ta băm theo tên, dùng hàm băm sau :

**HashValue = keyS0 + keyS1 + …+ keySL-1**

***Trong đó :***

* + keySi: là mã ASCII của kí tự S­i
  + L : độ dài của chuỗi

**Nhận xét :** Hai hàm băm trên đều xảy ra trường hợp đụng độ đó là : Với hai key tuy khác nhau nhưng lại cho kết quả băm giống nhau.

Để giải quyết vấn đề trên cần phải :

1. Xây dựng hàm băm mới
2. Sử dụng các phương pháp giải quyết các vấn đề đụng độ.

#### Xây dựng hàm băm mới

* + 1. Hàm băm Robert Sedgwicks (RS hash function)

Hàm băm dùng 2 số nguyên (a,b) để tính hashValue theo công thức sau:

**HashValue = HashValue\*a + keySi**

**a=a\*b ;**

Trong đó :

* a = 63689;
* b = 378551;
* keySi là mã ASCII của kí tự i trong chuỗi key

***Cài đặt:***

#include<bits/stdc++.h>

using namespace std;

string key;

unsigned int RSHash(string key)

{

unsigned int b = 378551;

unsigned int a = 63689;

unsigned int hashValue = 0;

unsigned int i = 0;

for (i = 0; i < key.size(); ++i)

{

hashValue = hashValue \* a + key[i];

a = a \* b;

}

return hashValue;

}

int main()

{

key="tinhoc";

cout<<RSHash(key);

}

**Lưu ý:** Sau khi đã hash thành công kết quả được % cho table\_size để đưa các key vào HashTable.

#### Hàm băm Polynomimal (Hàm băm đa thức)

Từng kí tự của chuỗi sẽ được chuyển đổi sang mã ASCII theo công thức sau:

**hashValue = keyS0\* aL-1 + keyS1\* aL-2 + …+ keySL-1\* a0**

***Trong đó:***

* + **keySi** là mã ASCII của kí tự i trong chuỗi **key**
  + L: độ dài của chuỗi
  + a là hằng số (thường là số nguyên tố)

***Cài đặt:***

#include<bits/stdc++.h>

using namespace std;

string key;

unsigned int polyHash(string key)

{

unsigned int HashValue=0;

unsigned int a=33;

for(int i = 0; i < key.size(); ++i)

{

HashValue = key[i] +a\* HashValue;

}

return HashValue;

}

int main()

{

key="tinhoc";

cout<<polyHash(key);

}

***Lưu ý:*** Sau khi đã hash thành công kết quả được % cho table\_size để đưa các key vào HashTable

#### Hàm băm Cyclic shift

Từng kí tự của chuỗi sẽ được chuyển đổi sang mã ASCII theo công thức sau:

**HashValue ^= ((HashValue<<a) + keySi + (HashValue>>b))**

Trong đó:

* + HashValue: ban đầu được khởi gán = 1,315,423,911
  + **keySi** là mã ASCII của kí tự i trong chuỗi **key**
  + a,b: được chọn có giá trị là số nguyên tố

Cài đặt:

#include<bits/stdc++.h>

using namespace std;

unsigned int CSHash(string key)

{

unsigned int hashValue = 1315423911;

unsigned int i = 0;

for (i = 0; i <key.size(); i++)

{

hashValue ^=((hashValue << 5) + key[i] +(hashValue >> 2));

}

return hashValue;

}

int main()

{

string key="tinhoc";

cout<<CSHash(key);

}

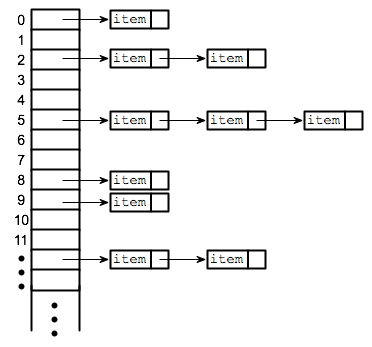
Sau khi đã hash thành công kết quả được % cho table\_size để đưa các key vào HashTable.

***Chú ý***: Mặc dù các hàm băm trên là một trong số các hàm băm đem lại kết quả tối ưu nhưng trong 1 số trường hợp cụ thể nó không phải các hàm băm hoàn hảo. Thực tế cho thấy bất kể hàm băm có tốt đến đâu, va chạm vẫn có thể xảy ra. Vì vậy, để duy trì hiệu suất của bảng băm, điều quan trọng là phải quản lý va chạm thông qua *các kỹ thuật giải quyết va chạm*.

#### Các phương pháp giải quyết va chạm

* + 1. Separate chaining (phương pháp kết nối)

Separate chaining là một kỹ thuật xử lý va chạm phổ biến nhất. Nó thường được cài đặt với danh sách liên kết. Để lưu giữ một phần tử trong bảng băm, ta phải thêm nó vào một danh sách liên kết ứng với chỉ mục của nó. Nếu có sự va chạm xảy ra, các phần tử đó sẽ nằm cùng trong 1 danh sách liên kết (xem ảnh để hiểu hơn).

.

Như vậy, kỹ thuật này sẽ đạt được tốc độ tìm kiếm **O(1)** trong trường hợp tối ưu và **O(N)** nếu tất cả các phần tử ở cùng 1 danh sách liên kết duy nhất. Đó là do có điều kiện 3 trong tiêu chí hàm băm tốt.

Cài đặt:

Giả định: Hàm băm sẽ trả về số int trong [0, 19].

vector <string> hashTable[20];

int hashTableSize=20;

**//Thêm vào bảng băm**

void insert(string s)

{

    // Compute the index using Hash Function

    int index = hashFunc(s);

    // Insert the element in the linked list at the particular index

    hashTable[index].push\_back(s);

}

**//Tìm kiếm**

void search(string s)

{

    //Compute the index by using the hash function

    int index = hashFunc(s);

    //Search the linked list at that specific index

    for(int i = 0;i < hashTable[index].size();i++)

    {

        if(hashTable[index][i] == s)

        {

            cout << s << " is found!" << endl;

            return;

        }

    }

    cout << s << " is not found!" << endl;

}

* + 1. Linear probing (dò tuyến tính)

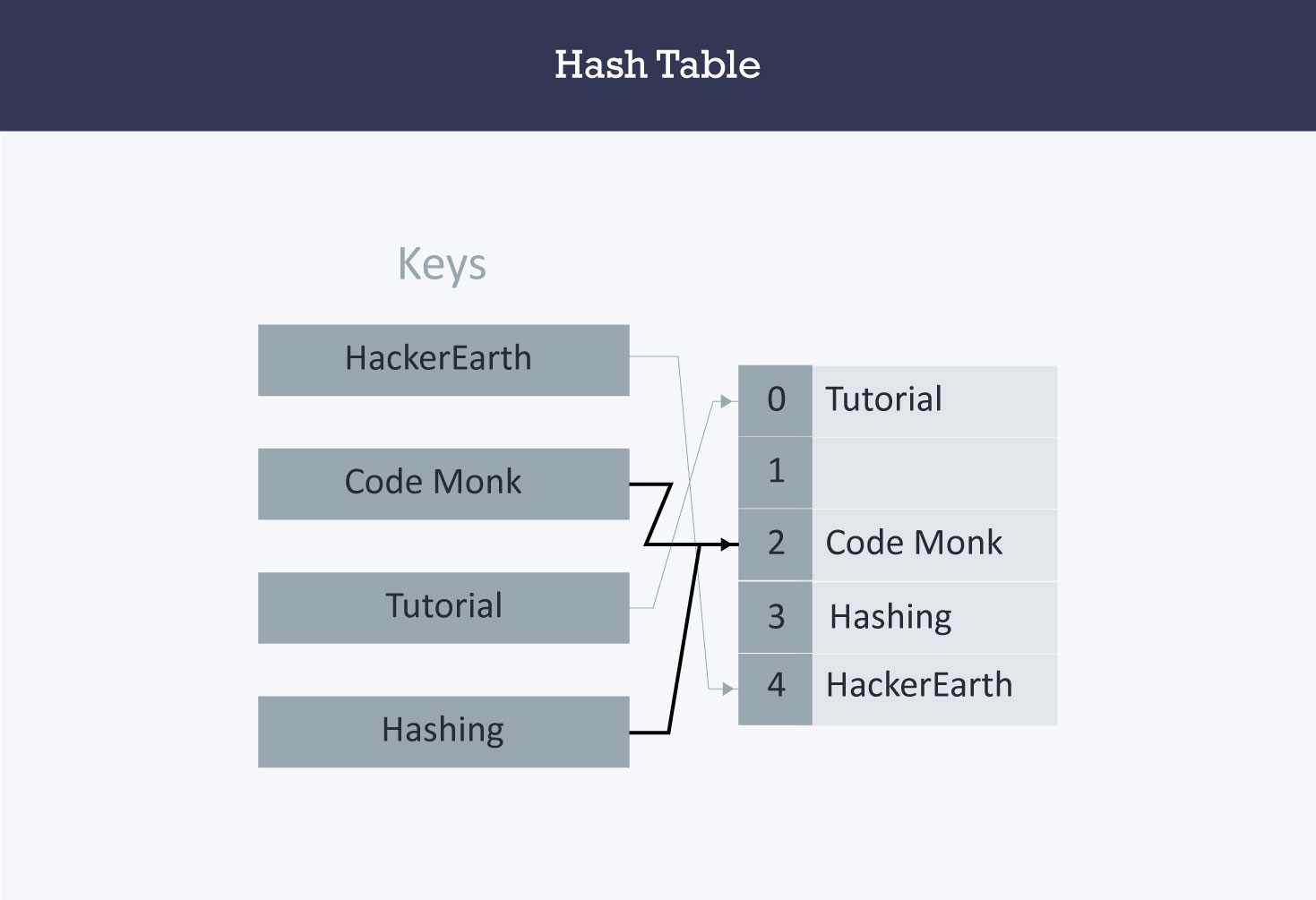
Trong kỹ thuật xử lý va chạm này, chúng ta sẽ không dùng linklist để lưu trữ mà chỉ có bản thân array đó thôi.

Khi thêm vào bảng băm, nếu chỉ mục đó đã có phần tử rồi; Giá trị chỉ mục sẽ được tính toán lại theo cơ chế tuần tự. Giả sử rằng chỉ mục là chỉ số của mảng, khi đó, việc tính toán chỉ mục cho phần tử được tính theo cách sau:

index = index % hashTableSize  
index = (index + 1) % hashTableSize  
index = (index + 2) % hashTableSize  
index = (index + 3) % hashTableSize

Và cứ thế theo cách như vậy chừng nào index thu được chưa có phần tử được sử dụng. Tất nhiên, không gian chỉ mục phải được đảm bảo để luôn có chỗ cho phần tử mới.

Như trong ví dụ dưới đây, chuỗi Hashing bị trùng chỉ mục (2) ở lần đầu tính chỉ mục. Do đó, nó được đẩy lên vị trí trống ở phía sau (3).



Cài đặt

Giả sử rằng:

* + Không có nhiều hơn 20 phần tử trong tập dữ liệu
  + Hàm băm sẽ trả về một số nguyên từ 0 đến 19
  + Tập dữ liệu phải là các phần tử duy nhất

string hashTable[21];

int hashTableSize = 21;

**\*Thêm vào bảng băm**

void insert(string s)

{

    //Compute the index using the hash function

    int index = hashFunc(s);

    //Search for an unused slot and if the index will exceed the hashTableSize then roll back

    while(hashTable[index] != "")

        index = (index + 1) % hashTableSize;

    hashTable[index] = s;

}

**\*Tìm kiếm**

void search(string s)

{

    //Compute the index using the hash function

    int index = hashFunc(s);

     //Search for an unused slot and if the index will exceed the hashTableSize then roll back

    while(hashTable[index] != s and hashTable[index] != "")

        index = (index + 1) % hashTableSize;

    //Check if the element is present in the hash table

    if(hashTable[index] == s)

        cout << s << " is found!" << endl;

    else

        cout << s << " is not found!" << endl;

}

* + 1. Quadratic Probing (dò bậc 2)

Ý tưởng cũng khá giống Linear Probing, nhưng cách tính chỉ mục có khác đôi chút:

index = index % hashTableSize  
index = (index + 12) % hashTableSize  
index = (index + 22) % hashTableSize  
index = (index + 32) % hashTableSize

Và cứ thế cho tới khi tìm được chỉ mục trống.

Cài đặt

Giả sử rằng:

* + Không có nhiều hơn 20 phần tử trong tập dữ liệu
  + Hàm băm sẽ trả về một số nguyên từ 0 đến 19
  + Tập dữ liệu phải là các phần tử duy nhất

string hashTable[21];

int hashTableSize = 21;

**Thêm phần tử**

void insert(string s)

{

    //Compute the index using the hash function

    int index = hashFunc(s);

    //Search for an unused slot and if the index will exceed the hashTableSize roll back

    int h = 1;

    while(hashTable[index] != "")

    {

        index = (index + h\*h) % hashTableSize;

             h++;

    }

    hashTable[index] = s;

}

**Tìm kiếm phần tử**

void search(string s)

{

    //Compute the index using the Hash Function

    int index = hashFunc(s);

     //Search for an unused slot and if the index will exceed the hashTableSize roll back

   int h = 1;

    while(hashTable[index] != s and hashTable[index] != "")

    {

        index = (index + h\*h) % hashTableSize;

             h++;

    }

    //Is the element present in the hash table

    if(hashTable[index] == s)

        cout << s << " is found!" << endl;

    else

        cout << s << " is not found!" << endl;

}

* + 1. Double hashing (Băm kép)

Vẫn giống 2 kỹ thuật ngay phía trên, chỉ khác ở công thức tính khi xảy ra va chạm như sau:

index = (index + 1 \* indexH) % hashTableSize;  
index = (index + 2 \* indexH) % hashTableSize;

Và cứ tiếp tục cho tới khi tìm được chỉ mục chưa được sử dụng.

Cài đặt

Giả sử rằng:

* + Không có nhiều hơn 20 phần tử trong tập dữ liệu
  + Hàm băm sẽ trả về một số nguyên từ 0 đến 19
  + Tập dữ liệu phải là các phần tử duy nhất

string hashTable[21];

int hashTableSize = 21;

**Thêm vào bảng băm**

void insert(string s)

{

    //Compute the index using the hash function1

    int index = hashFunc1(s);

    int indexH = hashFunc2(s);

    //Search for an unused slot and if the index exceeds the hashTableSize roll back

    while(hashTable[index] != "")

        index = (index + indexH) % hashTableSize;

    hashTable[index] = s;

}

**Tìm kiếm trên bảng băm**

void search(string s)

{

    //Compute the index using the hash function

    int index = hashFunc1(s);

    int indexH = hashFunc2(s);

     //Search for an unused slot and if the index exceeds the hashTableSize roll back

    while(hashTable[index] != s and hashTable[index] != "")

        index = (index + indexH) % hashTableSize;

    //Is the element present in the hash table

    if(hashTable[index] == s)

        cout << s << " is found!" << endl;

    else

        cout << s << " is not found!" << endl;

}

## BÀI TẬP ÁP DỤNG

Trong các bài toán thông thường, chúng ta chỉ cần sử dụng cấu trúc dữ liệu được cài đặt sẵn ở các ngôn ngữ lập trình: map, set trong C/C++, Java; Dictionary trong C#, Python. Đó chính là các bảng băm cực kỳ hữu dụng mà chúng ta vẫn hay sử dụng. Với các bài toán đặc thù, chúng ta sẽ phải tự viết cho mình hàm băm và xây dựng cấu trúc dữ liệu bảng băm cho phù hợp. Sau đây là một số bài tập ứng dụng:

1. Bài 1: Zero Quantity Maximization

(<https://codeforces.com/problemset/problem/1133/D>)

* 1. Đề bài

Cho hai mảng a,b. Mỗi mảng chứ n phần tử.

Hãy tạo ra một mảng c như sau : Chọn một số thực d (không cần là số nguyên), với mỗi i đặt ci = d.ai + bi (1≤i ≤n).

***Yêu cầu :*** Cần tăng tối đa số lượng các số 0 trong mảng c. Con số lớn nhất là gì nếu ta chọn d tối ưu?

***Input:*** File văn bản **ZQM.inp**

* + Dòng thứ nhất chứa một số nguyên n (1 ≤ n ≤ 2.105) – số phần tư trong mảng.
  + Dòng thứ hai chứa n số nguyên a1, a2, …, an (-109 ≤ai ≤109)
  + Dòng thứ ba chứ n số nguyên b1, b2, …, bn (-109 ≤bi ≤109).

***Output***: File văn bản **ZQM.out**

Một số nguyên là số lượng số 0 tối đa trong mảng c.

***Ví dụ:***

|  |  |  |
| --- | --- | --- |
| **ZQM.inp** | **ZQM.out** | **Giải thích** |
| 5  1 2 3 4 5  2 4 7 11 3 | 2 | Chọn d = - 2 |
| 3  13 37 39  1 2 3 | 2 | Chọn d = -1/13 |

* 1. Hướng dẫn giải thuật.

Ý tưởng để giải toán này đơn giản là tính số lần xuất hiện của bi/ai. Kết quả là lấy max số lần đó. Trường hợp ai, bi đồng thời bằng 0 thì cộng số trường hợp đó vào kết quả.

Vì giá trị -109 ≤ai ≤109 , ta dùng map để tạo bảng băm với index = bi/ai.

Độ phức tạp: O (N)

* 1. Chương trình.

#include<bits/stdc++.h>

using namespace std;

int main()

{

freopen("ZQM.inp","r",stdin);

freopen("ZQM.out","w",stdout);

map< long double,long int >m;

int n;

cin>>n;

long ar[n+10],br[n+10],ans=0,mx=0;

for(long i=0;i<n;i++)

cin>>ar[i];

for(long i=0;i<n;i++)

{

long b;

cin>>b;

if(ar[i]==0&&b==0)

ans+=1;

if(ar[i]!=0)

{

m[(b\*1.0)/ar[i]]++;

mx=max(mx,m[(b\*1.0)/ar[i]]);

}

}

cout<<mx+ans;

return 0;

}

* 1. Test : Tải theo link này

<https://drive.google.com/drive/folders/1LSlsFcL_MdHGnnq9jurIgDqgLxuHxlq7?usp=sharing>

1. Bài 2*:* Xâu con

(<https://vn.spoj.com/problems/SUBSTR>)

* 1. Đề bài

Cho xâu A và xâu B chỉ gồm các chữ cái thường. Xâu B được gọi là xuất hiện tại vị trí i của xâu A nếu: A[i] = B[1], A[i+1] = B[2], ..., A[i+length(B)-1] = B[length(B)].

***Yêu cầu:*** Hãy tìm tất cả các vị trí mà B xuất hiện trong A.

***Input:*** File văn bản substr.inp

* + Dòng 1: xâu A.
  + Dòng 2: xâu B.

Độ dài A, B không quá 1000000***.***

***Output:*** File văn bản substr.out

Ghi ra các vị trí tìm được trên 1 dòng (thứ tự tăng dần). Nếu B không xuất hiện trong A thì bỏ trắng.

***Ví dụ:***

|  |  |
| --- | --- |
| **Substr.inp** | **Substr.out** |
| aaaaa  aa | 1 2 3 4 |

* 1. Hướng dẫn thuật toán:

Gọi na là độ dài của của xâu A, nb là độ dài của xâu B.

Để xác định xâu B xuất hiện ở vị trí nào trong xâu A ta cần duyệt qua mọi vị trí i có thể trong A. Tức là cần so sánh xâu A[i..i+nb-1] có bằng xâu B không. Để dễ dàng cho việc so sánh ta cần sử dụng hàm băm Polynomial, tạo bảng băm hashValue cho mỗi xâu.

* + Tính hashB theo hàm băm Polynomial, với a=27, table\_size=109 + 7, L là độ dài của xâu B.
  + Tính hashA[i..j] theo cách tính sau:

hashA[i..j] = (hashA[j] –hashA[i-1]\*pow(a,j-i+1) +

table\_size\*table\_size)%table\_size;

* 1. Cài đặt:

#include<bits/stdc++.h>

using namespace std;

const long long table\_size=1000000000+7;

string A,B;

long long hashA[1000005],hashB,Pow[1000005];

int na,nb;

void khoitao()

{

cin>>A>>B;

na=A.size();nb=B.size();

A=" "+A;B=" "+B;

}

long long hashAij(int i,int j)

{

return (hashA[j] - hashA[i-1]\*Pow[j-i+1]+table\_size\*table\_size)%table\_size;

}

int main()

{

khoitao();

// tinh hashb

hashB=0;

for (int i=1;i<=nb;i++)

hashB=(hashB\*27+B[i])%table\_size;

// tinh hashA

hashA[0]=0;

for (int i=1;i<=na;i++)

hashA[i]=(hashA[i-1]\*27+A[i])%table\_size;

// tinh ham mu

Pow[0]=1;

for(int i=1;i<=na;i++)

Pow[i]=(Pow[i-1]\*27)%table\_size;

// tim vi tri xau B trong A

for(int i=1;i<=na-nb+1;i++)

{

// cout<<hashB<<" "<<hashAij(i,i+nb-1);

if (hashB==hashAij(i,i+nb-1)) cout<<i<<" ";

}

}

* 1. Test

<https://drive.google.com/drive/folders/1LSlsFcL_MdHGnnq9jurIgDqgLxuHxlq7?usp=sharing>

1. Bài 3: Xâu con đối xứng dài nhất

(<https://vn.spoj.com/problems/PALINY/>)

* 1. Đề bài

Một xâu được gọi là đối xứng nếu như khi đọc chuỗi này từ phải sang trái cũng thu được chuỗi ban đầu.

Cho xâu 𝑆 độ dài 𝑁 chỉ gồm các chữ cái latin in thường.

***Yêu cầu:*** Tìm xâu đối xứng dài nhất gồm các kí tự liên tiếp trong S, kết quả tìm được mod 109+7.

***Input:*** File văn bản **paliny.inp**

* + Dòng 1: N (độ dài của xâu S; N<=50 000)
  + Dòng 2: Xâu ký tự độ dài N

***Output:*** File văn bản **paliny.out**

Một dòng duy nhất gồm độ dài của xâu đối xứng dài nhất (mod 109+7)

***Ví dụ:***

|  |  |
| --- | --- |
| **Paliny.inp** | **Paliny.out** |
| 5  abacd | 3 |

***Giới hạn:***

Có 50% test ứng với N<=5000

50% test còn lại ứng với N>5000

* 1. Hướng dẫn giải thuật.

**Sub1:** Nếu 𝑁≤5000, Đặt 𝑓[𝑖][𝑗]=0/1 ứng với xâu con 𝑆𝑖…𝑗 không/có là xâu đối xứng.

Dễ nhận thấy, nếu tồn tại 1 dãy có độ dài là K (K>=2) thì luôn tồn tại 1 dãy có độ dài là K-2. vì thế, ta chặt nhị phân để tìm dãy có độ dài dài nhất.

Độ phức tạp : 𝑂(𝑁2)

**Sub2 :**

Thực hiện băm xâu S và xâu T là xâu đối xứng với xâu S

Với mỗi độ dài, ta sẽ kiểm tra trong xâu có tồn tại 1 xâu đối xứng độ dài như thế hay không. Chia 2 trường hợp : xâu S có độ dài chẵn hay lẻ.

Độ phức tạp : O(N\*log(N))

* 1. Chương trình.

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef pair<int,int> ii;

typedef unsigned long long ull;

#define FOR(i,l,r) for (int i=l;i<=r;i++)

#define FOD(i,r,l) for (int i=r;i>=l;i--)

const int N = 5e5+5, base = 1e9+9;

int n, ans;

ll p[N];

string s, t;

vector<ll> S, T;

vector<ll> getH(string s) {

vector<ll> h(s.size(),0);

h[0] = s[0]-'a'+1;

FOR(i,1,s.size()-1)

{

h[i] = (h[i-1] \* 27 + s[i]-'a'+1) % base;

}

return h;

}

ll getS(int l, int r) {

if (l == 0) return S[r];

ll ans = (S[r] - S[l-1] \* p[r-l+1] % base) % base;

while (ans < 0) ans += base;

return ans;

}

ll getT(int l, int r) {

if (l == 0) return T[r];

ll ans = (T[r] - T[l-1] \* p[r-l+1] % base) % base;

while (ans < 0) ans += base;

return ans;

}

bool check(int len) {

FOR(i,0,n-1-len+1) {

int j = i+len-1;

if (getS(i,j) == getT(n-j-1,n-i-1))

return true;

}

return false;

}

void solve(int L, int R, int k) {

while (L <= R) {

int mid = (L+R)/2, len = 2\*mid+k;

if (check(len))

ans = max(ans, len), L = mid+1;

else

R = mid-1;

}

}

int main() {

freopen("paliny.inp", "r", stdin);

freopen("paliny.out", "w", stdout);

scanf("%d\n", &n);

getline(cin, s);

t = s;

reverse(t.begin(), t.end());

S = getH(s);

T = getH(t);

p[0] = 1;

FOR(i,1,n-1) p[i] = p[i-1] \* 27 % base;

solve(0,(n-1)/2,1);

solve(1,n/2,0);

printf("%d\n", ans);

return 0;

}

* 1. Test

<https://drive.google.com/drive/folders/1LSlsFcL_MdHGnnq9jurIgDqgLxuHxlq7?usp=sharing>

1. Bài 4:Xâm nhập mật khẩu
   1. Đề bài

Gần đây, mạng xã hội *New Social Network* có sự xâm nhập thông tin người dùng. Mihael, một sinh viên trẻ, đã tìm thấy một lỗi xâm nhập tài khoản người dùng đó là: khi bạn nhập bất kỳ chuỗi ký tự có chứa chuỗi con bằng mật khẩu thực tế thì đăng nhập sẽ thành công. Ví dụ: nếu người dùng có mật khẩu abc thì khi nhập một trong các chuỗi abc, abcd hoặc xyaabccz, hệ thống sẽ đăng nhập thành công.

***Yêu cầu***: Mihael muốn biết có bao nhiêu cặp người dùng khác nhau sao cho người dùng đầu tiên sử dụng mật khẩu riêng của họ, có thể đăng nhập như người dùng thứ hai. Bạn hãy giúp Mihael tính nhanh điều đó.

***Input****:* File văn bản PASS.INP gồm:

* + Dòng đầu tiên chứa số nguyên dương N (1≤ N ≤20000) - số lượng người dùng.
  + N dòng sau chứa mật khẩu của người dùng. Các mật khẩu bao gồm ít nhất một hoặc nhiều nhất là 10 chữ cái viết thường của bảng chữ cái tiếng Anh.

***Output***: File văn bản **PASS.OUT** gồm:

Một dòng duy nhất chứa số lượng cặp người dùng theo yêu cầu nói trên.

***Ví dụ:***

|  |  |  |
| --- | --- | --- |
| **PASS.INP** | **PASS.OUT** | ***Giải thích*** |
| 3  a  b  ab | 2 | *Gồm 2 cặp đó là:*  *(3,1); (3,2)* |
| 4  a  ab  a  abc | 7 | *Gồm 7 cặp người dùng (1,3);(2,1); (2,3); (3,1);*  *(4,1); (4,2); (4,3).* |

***Giới hạn:***

* + *Có 40% số test ứng với 40% số điểm có N≤ 2000*
  + *Có 60% số test ứng với 60% số điểm có N≤ 2x104*
  1. Hướng dẫn giải thuật

**Sub 1**

Với mỗi mật khẩu X[i] của người dùng thứ i, cần trả lời truy vấn có bao nhiêu mật khẩu khác chứa X[i] dưới dạng chuỗi con?

Đối với mỗi mật khẩu, chúng ta giải quyết các truy vấn bằng cách lặp lại tất cả các mật khẩu và kiểm tra mỗi chuỗi con có thể. Độ phức tạp tính toán là O(N2\* L2), trong đó N là số lượng người dùng và L là độ dài mật khẩu.

Như vậy với cách này chỉ có thể giải quyết được sub1 ứng 40% test của bài có N≤2000.

**Sub 2**

Có thể tăng tốc bằng cách sử dụng bảng băm.

* + Với mỗi mật khẩu có các chuỗi con khác nhau là gì. Ứng với mỗi chuỗi s ta cần băm các chuỗi con đó.
  + Để trả lời các truy vấn có bao nhiêu mật khẩu khác chứa mật khẩu người dùng thứ i làm chuỗi con thì ta dùng **map** để đếm số lần chuỗi con xuất hiện. Vì các kí tự được lặp lại nhiều lần trong 1 chuỗi nên cần phải lưu ý khi đếm số lần xuất hiện của chuỗi con.
  + Kết quả là tổng số lần xuất hiện của các chuỗi s đã cho trừ đi n.

Độ phức tạp là: O (N\*L2).

* 1. Chương trình.

#include <bits/stdc++.h>

using namespace std;

int n,i,j,k,ans;

map < long long , int > mp;

string s;

long long HAS,A[200002];

int main() {

freopen("pass.inp","r",stdin);

freopen("pass.out","w",stdout);

cin>>n;

for (i=1;i<=n;i++) {

cin>>s;

vector < long long > all;

for (j=0;j<s.size();j++) {

HAS=0;

for (k=j;k<s.size();k++) {

HAS=HAS\*27+s[k]-'a'+1;

all.push\_back(HAS);

}

if (j == 0)

{

A[i]=HAS;

}

}

sort(all.begin(),all.end());

mp[all[0]]++;//cout<<all.size();

for (j=1;j<all.size();j++)

if (all[j] != all[j-1])

mp[all[j]]++;

}

for (i=1;i<=n;i++)

ans+=mp[A[i]];

cout<<ans-n<<endl;

}

### Test

<https://drive.google.com/drive/folders/1LSlsFcL_MdHGnnq9jurIgDqgLxuHxlq7?usp=sharing>

1. Bài 5: Chuỗi abc

<https://codeforces.com/problemset/problem/514/C>

* 1. Đề bài

Cho tập X gồm n xâu. Có m truy vấn.

***Yêu cầu***: Với mỗi truy vấn: “Cho một xâu *s*, xác định xem trong tập *X* có tồn tại xâu *t* sao cho xâu *t* có cùng số kí tự với xâu *s* nhưng sai khác với xâu *s* một kí tự ở một vị trí xác định”.

Viết chương trình thực hiện yêu cầu trên.

***Input***: File văn bản abc.inp

* + Dòng đầu tiên chứa hai số không âm n và m (0 ≤ n ≤ 3·105, 0 ≤ m ≤ 3·105) là số lượng các chuỗi ban đầu và số lượng truy vấn.
  + *N* dòng tiếp theo chứa các xâu không rỗng trong tập *X*.
  + *M* dòng tiếp theo chứa các xâu không rỗng là các truy vấn của bài toán.

Biết rằng tổng chiều dài của các dòng dữ liệu vào không vượt quá 6x105. Mỗi dòng chỉ chứa các kí tự *‘a’,’b’,’c’*.

***Output***: file văn bản abc.out

Với mỗi truy vấn in trên một dòng: ghi “YES” nếu có tồn tại xâu *t* theo yêu cầu, ngược lại ghi “NO”.

Ví dụ:

|  |  |
| --- | --- |
| ABC.INP | ABC.OUT |
| 2 3 aaaaa acacaca aabaa ccacacc caaac | YES  NO  NO |

* 1. Hướng dẫn giải thuật:
  + Sử dụng hàm băm Polynomial, tạo bảng băm hashValue cho mỗi chuỗi trong n chuỗi trong tập X. Với mỗi chuỗi t trong m chuỗi truy vấn, ta sẽ tìm tất cả các chuỗi thu được bằng cách thay thế 1 kí tự của chuỗi t bằng 1 kí tự khác. Chọn các tham số trong hàm băm Polynimial: a = 257, table\_size = 109 + 7, L = 1000001.
  + Lấy hashValue của tất cả các chuỗi vừa tạo, so sánh với hashValue của bảng băm n chuỗi trên. Nếu chuỗi đó xuất hiện trong bảng băm thì in ra YES, ngược lại NO.

Độ phức tạp: O (L\*log(n)) với L là tổng độ dài của tất cả các chuỗi, log(n) là độ phức tạp tìm kiếm trong bảng băm.

* 1. Cài đặt

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int L = 1000001;

const ll table\_size = 1e9 + 7;

const ll a = 257;

int n, m;

string keys, t, tmp;

ll f[L];

set<ll> dic;

void init()

{

f[0] = 1;

for (int i = 1; i < L; i++)

f[i] = (f[i - 1] \* a) % table\_size;

}

ll polyHash(string keys)

{

ll hashValue = 0;

for (int i = 0; i < keys.size(); i++)

hashValue = (hashValue \* a + keys[i]) % table\_size;

return hashValue;

}

bool check(string t)

{

ll h = polyHash(t);

ll len = t.length();

for (int i = 0; i < t.size(); i++)

{

for (ll c = 'a'; c <= 'c'; c++)

{

if (c == t[i])

continue;

ll new\_hashValue = ((((c - t[i]) \* f[len - i - 1]) % table\_size + table\_size) + h) % table\_size;

if (dic.find(new\_hashValue) != dic.end())

return true;

}

}

return false;

}

int main()

{

freopen("abc.inp","r",stdin);

freopen("abc.out","w",stdout);

init();

cin >> n >> m;

dic.clear();

for (int i = 0; i < n; i++)

{

cin >> keys; // co n chuoi keys

dic.insert(polyHash(keys)); // băm n chuoi

}

for (int i = 0; i < m; i++)

{

cin >> t;

if (check(t))

cout << "YES" << endl;

else

cout << "NO" << endl;

}

return 0;

}

* 1. Test

<https://drive.google.com/drive/folders/1LSlsFcL_MdHGnnq9jurIgDqgLxuHxlq7?usp=sharing>

1. Một vài bài toán khác

<https://vn.spoj.com/problems/C11STR2/>

<https://codeforces.com/problemset/problem/4/C>

<https://codeforces.com/problemset/problem/28/D>

<https://codeforces.com/problemset/problem/25/E>

<https://codeforces.com/problemset/problem/30/E>

# PHẦN KẾT LUẬN

Trong chuyên đề, tôi đã trình bày về khái niệm cấu trúc dữ liệu băm, các xây dựng các hàm băm để ứng dụng để giải một số bài toán trong Tin học giúp giảm độ phức tạp của bài toán.

Sau khi áp dụng cấu trúc dữ liệu băm vào một số bài toán học sinh giỏi, đặc biệt là các bài toán xử lí xâu, tôi thấy nó mang lại hiệu quả rất rõ rệt. Bây giờ chúng ta đã có một phương pháp đơn giản, dễ dàng hơn để thực hiện. Do thời gian còn hạn chế và kiến thức còn chưa được sâu, rộng nên chắc chắn chuyên đề còn nhiều thiếu sót. Tôi rất mong quý thầy cô đồng nghiệp đóng góp ý kiến để chuyên đề được hoàn thiện hơn.

# TÀI LIỆU THAM KHẢO

1. Tài liệu giáo khoa chuyên Tin tập 1, 2, 3.
2. Website: <http://vn.spoj.com/>
3. Website: <http://codeforces.com>
4. Website: <https://vi.wikipedia.org/wiki/H%C3%A0m_b%C4%83m>
5. Website: <https://nguyenvanhieu.vn/bang-bam-hash-tables/>
6. Website: <http://www.partow.net/programming/hashfunctions/index.html>