

TÊN MÔN HỌC : Internet và giao thức

NHÓM HỌC PHẦN : 14

TÊN ĐỀ TÀI NHÓM : Bảng băm và ứng dụng

NHÓM BÀI TẬP LỚN : Nhóm 8

TÊN THÀNH VIÊN NHÓM : Bùi Hải An B18DCAT001

Nguyễn Bảo Long B18DCVT257

Hoàng Nguyễn Long Thảo B18DCDT238

Nguyễn Quốc Việt B18DCVT441

**MỤC LỤC**

[GIỚI THIỆU CHUNG VỀ VẤN ĐỀ NGHIÊN CỨU 1](#_Toc100872219)

[1. KHÁI NIỆM 2](#_Toc100872220)

[2. BẢNG BĂM (HASH TABLE) 4](#_Toc100872221)

[2.1. Giải quyết xung đột bằng kĩ thuật dây chuyền 4](#_Toc100872222)

[2.2. Phân tích kĩ thuật băm bằng dây chuyền 5](#_Toc100872223)

[2.3. Băm trong cấu trúc dữ liệu 7](#_Toc100872224)

[2.4. Cơ chế băm 7](#_Toc100872225)

[2.5. Giá trị khóa băm 7](#_Toc100872226)

[3. HÀM BĂM (HASH FUNCTION) 8](#_Toc100872227)

[3.1. Phương pháp chia 9](#_Toc100872228)

[3.2. Phương pháp nhân 10](#_Toc100872229)

[3.3. Kĩ thuật phổ băm 11](#_Toc100872230)

[3.4. Kĩ thuật băm hoàn hảo 14](#_Toc100872231)

[CHƯƠNG II: ỨNG DỤNG BẢNG BĂM 19](#_Toc100872232)

[1. CHE DẤU DỮ LIỆU GỐC 19](#_Toc100872233)

[2. HASHING TRONG ĐỊNH DANH DỮ LIỆU 20](#_Toc100872234)

[3. NÉN DỮ LIỆU 22](#_Toc100872235)

[4. ỨNG DỤNG TRONG LẬP CHỈ MỤC VÀ TRUY XUẤT DỮ LIỆU 22](#_Toc100872236)

[5. ỨNG DỤNG TRONG BẰNG CHỨNG CÔNG VIỆC (PROOF OF WORK) 24](#_Toc100872237)

# PHÂN CÔNG CÔNG VIỆC

|  |  |
| --- | --- |
| **TÊN THÀNH VIÊN** | **CÔNG VIỆC** |
| Bùi Hải An | Tìm hiểu về bảng băm : cơ chế băm, giá trị khóa băm, băm trong cấu trúc dữ liệu  Ứng dụng : lập chỉ mục và truy xuất dữ liệu |
| Nguyễn Bảo Long | Tìm hiểu về hàm băm : phương pháp nhân, phương pháp chia,  Ứng dụng : nén dữ liệu , Proof of Work |
| Nguyễn Quốc Việt | Làm phần giới thiệu chung về vấn đề nghiên cứu  Tìm hiểu về hàm băm : kỹ thuật về phổ băm |
| Hoàng Nguyễn Long Thảo | Ứng dụng : che dấu dữ liệu gốc, hashing trong định danh dữ liệu |

# GIỚI THIỆU CHUNG VỀ VẤN ĐỀ NGHIÊN CỨU

Trong khoa học máy tính, bảng băm là một cấu trúc dữ liệu sử dụng hàm băm để ánh xạ từ giá trị xác định, được gọi là khóa (ví dụ như tên của một người), đến giá trị tương ứng (ví dụ như số điện thoại của họ). Do đó, bảng băm là một mảng kết hợp. Hàm băm được sử dụng để chuyển đổi từ khóa thành chỉ số (giá trị băm) trong mảng lưu trữ các giá trị tìm kiếm.

Trong trường hợp lý tưởng, hàm băm luôn chuyển đổi các khóa khác nhau đến các chỉ số khác nhau. Tuy nhiên trong thực tế, điều này hiếm khi xảy ra (trừ khi các khóa là cố định: không có thêm khóa mới nào được bổ sung vào bảng sau khi tạo bảng). Thay vào đó, hầu hết các thiết kế bảng băm đều giả sử các khóa khác nhau có thể được băm vào cùng một giá trị (gọi là va chạm băm), và cung cấp cách giải quyết va chạm.

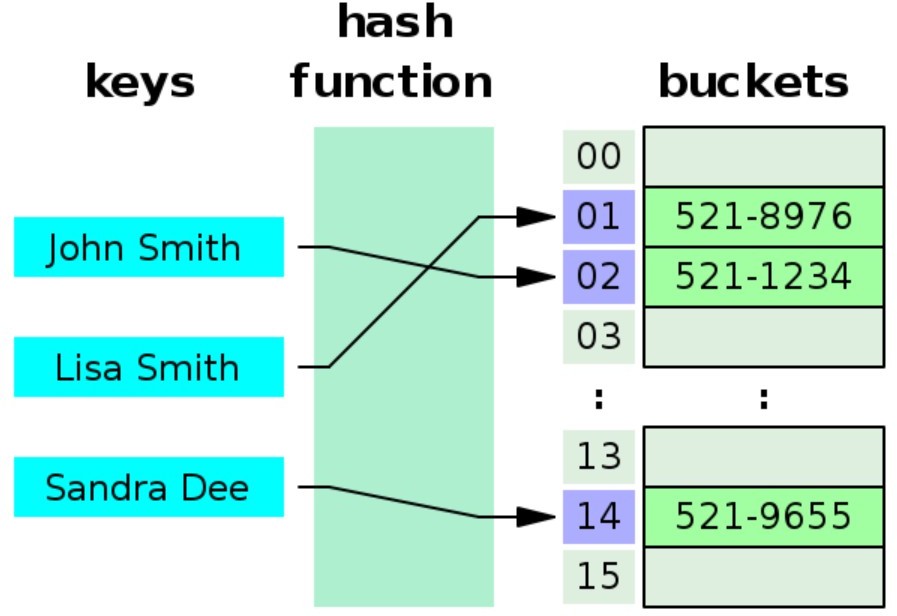
Trong một bảng băm có kích thước đủ lớn, thời gian trung bình cho mỗi lần tra cứu là độc lập với số lượng phần tử trong bảng. Nhiều thiết kế bảng băm cũng cho phép chèn thêm và xóa bỏ tùy ý của cặp khóa-giá trị trong thời gian trung bình (hoặc trừ dần) không đổi cho mỗi thao tác.

Trong nhiều trường hợp, bảng băm có hiệu quả hơn so với cây tìm kiếm hoặc bất kỳ cấu trúc dữ liệu tìm kiếm nào. Vì lý do này, chúng được sử dụng rộng rãi trong nhiều loại phần mềm máy tính, đặc biệt là cho mảng kết hợp, lập chỉ mục cơ sở dữ liệu, tổ chức bộ nhớ đệm, và cấu trúc dữ liệu tập hợp. Dưới đây chúng ta sẽ tìm hiểu sâu hơn về bảng băm, hàm băm, các phương pháp tính toán ở trong hàm băm và ứng dụng của bảng băm trong thực tế

**CHƯƠNG I: TỔNG QUAN VỀ BẢNG BĂM**

## KHÁI NIỆM

* Băm là một kỹ thuật được sử dụng để định danh một đối tượng cụ thể trong một nhóm các đối tượng tương tự. Một bảng băm (hash table) là một cấu trúc dữ liệu ánh xạ giữa khóa và giá trị, tức là tương ứng với một khóa bảng băm sẽ trả về một giá trị.
* Ví dụ về việc sử dụng bảng băm:
* 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 một mã số riêng và mã số đó có thể được dùng để xác định các thông tin của sách.
* Trong cả 2 ví dụ trên, các sinh viên và các cuốn sách được “băm” thành các mã số duy nhất (không trùng lặp nhau).
* Giả sử có một đối tượng và muốn gán cho nó một cái khóa (key) để giúp tìm kiếm dễ dàng hơn.
* Để lưu giữ cặp <khóa, giá trị>(<key, value>), ta 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 sẽ cần tới “băm”(hashing).



* Hiển nhiên, việc sử dụng bảng địa chỉ trực tiếp sẽ gặp khó khăn vì khi không gian *U* lớn thì việc lưu trữ một bảng *T* có kích cỡ |*U*| có thể không thực hiện được. Hơn nữa, tập hợp *K* các khóa được lưu trữ thực tế có thể nhỏ so với *U* đến mức hầu hết không gian phân bổ cho *T* sẽ bị lãng phí.
* Khi tập hợp *K* các khóa lưu trữ trong một từ điển nhỏ hơn nhiều so với không gian *U* tất cả các khóa có thể, một bảng băm sẽ yêu cầu không gian lưu trữ ít hơn nhiều so với bảng địa chỉ trực tiếp. Cụ thể, các yêu cầu về không gian lưu trữ có thể giảm tới mức Θ(|*K*|), trong khi việc tìm kiếm một thành phần trong bảng băm vẫn chỉ có độ phức tạp là *O*(1). Hạn chế duy nhất với bảng băm là ta áp dụng cho trường hợp trung bình, trong khi đó bảng địa chỉ trực tiếp áp dụng cho trường hợp xấu nhất.
* Với kỹ thuật định địa chỉ trực tiếp, một thành phần có khóa *k* được lưu trữ trong khe *k*. Với kỹ thuật bảng băm, thành phần này được lưu trữ trong khe *h(k)*; nghĩa là một hàm băm (**hash function**) *h* được dùng để tính toán giá trị lưu tại khe dựa vào khóa *k*. Ở đây *h* ánh xạ (map) các khóa trong không gian *U* vào các khe của một bảng băm *T*[0..m-1], với kích thước *m*:

*h*: U → {0, 1, …, m-1}

* Ta nói rằng một thành phần có khóa *k* sẽ băm (**hash**) theo khe *h(k)*, hay *h(k)* là giá trị băm (**hash value**) của khóa *k*. **Hình 2** minh họa khái niệm cơ bản. Cốt lõi của của hàm băm là rút gọn miền các chỉ số mảng cần được quản lý. Thay vì |*U*| giá trị ta chỉ cần quản lý *m* (|*U*| > *m*) giá trị. Nhờ đó, yêu cầu không gian lưu trữ cũng sẽ được giảm tương ứng.

  
***Hình 2.*** *Dùng một hàm băm h để ánh xạ các khóa theo các khe của bảng băm. Các khóa k1 và k5 ánh xạ theo cùng khe nên chúng xung đột.*

* Điểm cần lưu ý ở đây là hai khóa có thể được băm theo cùng khe. Chúng ta gọi đó là sự xung đột (**collision**). Và ta cũng có các kỹ thuật để giải quyết sự xung đột này một cách hiệu quả.
* Tất nhiên, giải pháp lý tưởng là tránh các xung đột. Ta có thể cố gắng đạt được mục tiêu này bằng cách chọn ra một hàm băm *h* thích hợp. Có một ý tưởng cho rằng nên để *h* xuất hiện “ngẫu nhiên”, nhờ đó tránh được sự xung đột hay ít ra cũng giảm được số lần xung đột của các giá trị băm. (Thuật ngữ “to hash” (băm) gợi lên hình ảnh băm và trộn một cách ngẫu nhiên đã nêu được tư tưởng của phương pháp này). Tuy nhiên, do |*U*| > *m*, nên chắc chắn sẽ có 2 khóa có cùng giá trị băm; do đó việc tránh hẳn các xung đột là không thể. Như vậy, tuy hàm băm có vẻ “ngẫu nhiên” và đuợc thiết kế kỹ lưỡng để có thể giảm thiểu số lần xung đột, song ta vẫn phải dùng một phương pháp để giải quyết xung đột xảy ra.
* Sau đây, chúng ta sẽ khảo sát kỹ thuật giải quyết xung đột đơn giản nhất, có tên dây chuyền (còn gọi là phương pháp băm mở). Phần 2.4 sẽ tiếp tục giới thiệu một phương pháp giải quyết xung đột khác có tên định địa chỉ mở (còn gọi là phương pháp băm đóng).

## BẢNG BĂM (HASH TABLE)

## Giải quyết xung đột bằng kĩ thuật dây chuyền

* + - Trong kỹ thuật dây chuyền, ta đặt tất cả các thành phần băm theo cùng khe vào trong một danh sách liên kết như trong **Hình 3**. Khe *j* chứa 1 biến trỏ đến đầu của danh sách chứa tất cả các thành phần đã lưu trữ băm theo *j*; nếu không có thành phần nào như vậy, khe *j* chứa NIL.



***Hình 3****. Giải quyết xung đột bằng dây chuyền. Mỗi khe bảng băm T[j] chứa 1 danh sách liên kết bao gồm tất cả các khóa có giá trị băm là j*

* Các phép toán từ điển trên một bảng băm *T* thường dễ thực thi khi các xung đột được giải quyết bằng kỹ thuật dây chuyền.
* CHAINED-HASH-INSERT(*T*, *x*)

*Chèn x vào đầu danh sách T[h(key[x])]*

* CHAINED-HASH-SEARCH(*T*, *k*)

*Tìm kiếm một phần tử với khóa k trong danh sách T[h(k)]*

* CHAINED-HASH-DELETE(*T*, *x*)

*Xóa x khỏi danh sách T[h(key[x])]*

* Thời gian thực hiện trong trường hợp xấu nhất của thủ tục chèn là *O*(1). Với thủ tục tìm kiếm, thời gian thực hiện trong trường hợp xấu nhất tỷ lệ với chiều dài danh sách; chúng ta sẽ phân tích thủ tục này kỹ hơn trong phần sau. Việc xóa 1 thành phần *x* có thể hoàn thành trong *O*(1) thời gian nếu danh sách liên kết đôi. Nếu danh sách liên kết đơn, thủ tục xóa và tìm kiếm có cùng thời gian thực hiện.

## Phân tích kĩ thuật băm bằng dây chuyền

* Kỹ thuật băm bằng dây chuyền thực hiện tốt tới mức nào? Cụ thể hơn, phải mất bao lâu để tìm kiếm một phần tử với một khóa đã cho?
* Cho một bảng băm *T* có *m* khe lưu trữ *n* thành phần, ta định nghĩa hệ số tải (**load factor**) của *T* là *α* = *n*/*m*, và ta gọi đây là số trung bình các phần tử được lưu trữ trong một danh sách. Phân tích của chúng ta sẽ dựa theo các giá trị của α, đó là lớn hơn, bằng hoặc nhỏ hơn 1.
* Trong trường hợp xấu nhất, cách xử lý của kỹ thuật băm bằng dây chuyền là rất tệ: tất cả n khóa đều băm theo cùng 1 khe, tạo ra danh sách liên kết có chiều dài n. Như vậy, thời gian để tìm kiếm trong trường hợp xấu nhất là *O*(n) cộng thời gian để tính toán hàm băm - không tốt hơn so với việc dùng 1 danh sách liên kết cho tất cả các thành phần. Rõ ràng, các bảng băm không được dùng để thực hiện cho trường hợp xấu nhất của chúng.
* Khả năng thực hiện trung bình của kỹ thuật băm tùy thuộc vào mức độ tốt xấu khi hàm băm *h* phân phối tập hợp các khóa được lưu trữ giữa *m* khe, tính trung bình. Phần 2.3 sẽ giải thích chi tiết hơn vấn đề này. Ở đây, ta sẽ giả định rằng mọi phần tử đã cho đều có khả năng như nhau để băm vào bất kỳ khe nào trong số *m* khe, và chúng không phụ thuộc vào nơi mà bất kỳ phần tử nào khác đã được băm đến. Ta gọi đây là giả thiết của kỹ thuật băm đều **(simple uniform hashing)**
* Ta giả định rằng giá trị băm *h(k)* có thể được tính toán trong thời gian *O*(1), sao cho thời gian cần thiết để tìm kiếm một phần tử có khóa *k* tùy thuộc vào chiều dài của danh sách *T*[*h(k)*]. Tạm thời bỏ qua thời gian *O*(1) cần thiết để tính toán hàm băm và truy cập khe *h(k)*, ta tập trung vào số phần tử dự trù mà thuật toán tìm kiếm sẽ xem xét, nghĩa là, số lượng các phần tử trong danh sách *T*[*h(k)*] được kiểm tra để xem các khóa của chúng có bằng *k* hay không. Ta sẽ xét 2 trường hợp. Đầu tiên, đợt tìm kiếm không thành công: không có phần tử nào trong bảng có khóa *k*. Thứ hai, đợt tìm kiếm thành công: tìm được phần tử có khóa k.
* **Định lý 1**

Trong một bảng băm với các xung đột được giải quyết bằng kỹ thuật dây chuyền, một đợt tìm kiếm không thành công bình quân sẽ mất thời gian *O*(1+α), dưới giả thiết của kỹ thuật băm đều.

* **Chứng minh**

Dưới giả thiết của kỹ thuật băm đều, mọi khóa *k* chưa được lưu trong bảng đều có khả năng như nhau để băm theo bất kỳ khe nào trong số *m* khe. Như vậy, thời gian trung bình để tìm không thành công một khóa *k* là thời gian trung bình để tìm đến cuối của một trong *m* danh sách. Chiều dài trung bình của một danh sách như vậy là hệ số tải *α*=*n*/*m*. Như vậy, số thành phần dự trù được xem xét trong đợt tìm kiếm không thành công là *α*, tổng thời gian cần thiết (kể cả thời gian để tính toán *h(k)*) là *O*(1+α).

* **Định lý 2**

Trong một bảng băm với các xung đột được giải quyết bằng kỹ thuật dây chuyền, một đợt tìm kiếm thành công bình quân sẽ mất thời gian *O*(1+ α), dưới giả thiết của kỹ thuật băm đều.

* **Chứng minh**
  + Ta giả định rằng phần tử đang được tìm kiếm có khả năng như nhau để bằng bất kỳ phần tử nào trong *n* phần tử được lưu trữ trong bảng. Ta cũng giả định rằng thủ tục CHAINED-HASH-INSERT chèn một phần tử mới ở cuối thay vì ở đầu danh sách.
  + Số phần tử được xem xét trong một đợt tìm kiếm thành công phần tử *x* là nhiều hơn 1 so với số lượng các phần tử xuất hiện trong danh sách trước khi *x* được chèn vào đó.
  + Do đó, để tìm ra số phần tử dự kiến được xem xét, ta lấy trung bình, trên *n* phần tử trong bảng, của 1 cộng với chiều dài dự kiến của danh sách đó sẽ là *(i-1)*/*m*, và như vậy số phần tử dự kiến được xem xét trong 1 đợt tìm kiếm thành công là:







* Như vậy, tổng thời gian cần thiết cho một đợt tìm kiếm thành công (kể cả thời gian tính toán hàm băm) là *O*(2+ α/2 – 1/2m) = *O*(1+α)
* Kỹ thuật phân tích này có ý nghĩa gì? Nếu số lượng các khe của bảng băm ít nhất tỷ lệ với số lượng các thành phần trong bảng, ta có *n* = *O*(*m*), lúc đó *α* = *n*/*m* = *O*(*m)*/*m* = *O*(1). Như vậy, tính trung bình việc tìm kiếm tốn lượng thời gian bất biến. Do đó, việc chèn mất thời gian *O*(1) và việc xóa mất thời gian *O*(1) trong trường hợp xấu nhất khi danh sách thuộc dạng danh sách liên kết đôi. Tất cả các phép toán từ điển có thể được thực hiện trong thời gian *O*(1), tính trung bình.

## Băm trong cấu trúc dữ liệu

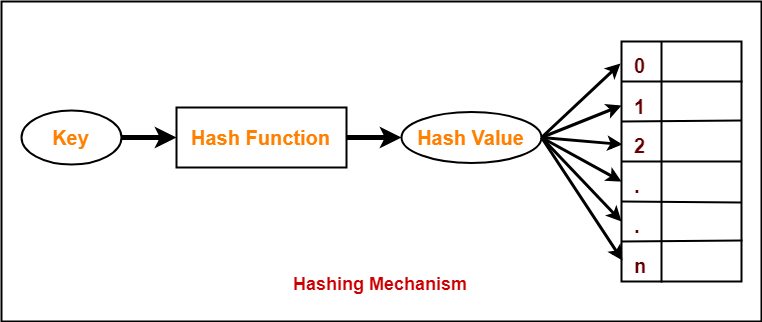
* Hashing là một kỹ thuật nổi tiếng để tìm kiếm bất kỳ phần tử cụ thể nào trong số một số phần tử.
* Nó giảm thiểu số lượng so sánh trong khi thực hiện tìm kiếm.
* Thuận lợi : không giống như các kỹ thuật tìm kiếm khác
* Hashing cực kỳ hiệu quả.
* Thời gian nó thực hiện tìm kiếm không phụ thuộc vào tổng số phần tử.
* Nó hoàn thành việc tìm kiếm với độ phức tạp thời gian không đổi O (1).

## Cơ chế băm

* Một cấu trúc dữ liệu mảng được gọi là bảng Hash được sử dụng để lưu trữ các mục dữ liệu.
* Dựa trên giá trị khóa băm, các mục dữ liệu được chèn vào bảng băm.

## Giá trị khóa băm

* Giá trị khóa băm là một giá trị đặc biệt dùng làm chỉ mục cho một mục dữ liệu.
* Nó chỉ ra nơi mục dữ liệu nên được lưu trữ trong bảng băm.
* Giá trị khóa băm được tạo bằng hàm băm.



## HÀM BĂM (HASH FUNCTION)

* Trong phần này, chúng ta đề cập vài vấn đề liên quan đến thiết kế các hàm băm tốt và sau đó trình bày ba lược đồ để tạo chúng: kỹ thuật băm bằng phép chia, kỹ thuật băm bằng phép nhân và kỹ thuật phổ băm (universal hashing).

**Điều gì tạo nên một hàm băm tốt?**

* Một hàm băm tốt thỏa (xấp xỉ) giả thiết của kỹ thuật băm đều đơn giản: mỗi khóa có khả năng như nhau để băm vào bất kỳ vị trí nào trong số *m* khe. Chính thức hơn, ta hãy mặc nhận rằng mỗi khóa được rút ra độc lập từ *U* theo phân phối xác suất *P*; nghĩa là, *P*(*k*) là xác xuất mà k được rút. Như vậy giả thiết của kỹ thuật băm đều đơn giản là:

j = 0, 1, …, m-1 (1)

* Đáng tiếc, ta thường không thể kiểm tra điều kiện này, bởi *P* thường chưa biết.
* Đôi lúc ta biết được phép phân phối *P*. Ví dụ, giả sử các khóa được xem là các số thực ngẫu nhiên *k* được phân phối độc lập và đồng đều trong miền giá trị 0 ≤ k < 1. Trong trường hợp này, hàm băm:

*h*(*k*) = ⌊*km*⌋ có thể được chứng tỏ thỏa phương trình (1)

* Trong thực tế, có thể dùng các kỹ thuật phỏng đoán (heuristic) để tạo một hàm băm có khả năng thực hiện tốt. Thông tin định tính về *P* đôi lúc tỏ ra hữu ích trong tiến trình thiết kế này. Ví dụ, hãy xét bảng ký hiệu của một bộ biên dịch, ở đó các khóa là các chuỗi ký tự tùy ý biểu thị cho các dấu định danh (identifiers) trong một chương trình. Nói chung, các ký hiệu liên quan mật thiết, như *pt* và *pts*, thường xảy ra trong cùng chương trình. Một hàm băm tốt sẽ giảm thiểu cơ hội các biến thể được băm theo cùng khe.
* Một cách tiếp cận chung đó là suy ra giá trị băm theo cách được dự trù là độc lập với bất kỳ khuôn mẫu nào đó có thể tồn tại trong dữ liệu. Ví dụ, “phương pháp chia” (được thảo luận trong phần 2.3.1) tính toán giá trị băm như là số dư khi khóa được chia cho phần số nguyên tố định rõ. Trừ phi số nguyên tố này bằng cách nào đó có liên quan đến các khuôn mẫu trong phép phân phối xác xuất *P*, phương pháp này cho các kết quả tốt.
* Cuối cùng, cần lưu ý có vài ứng dụng của các hàm băm có thể yêu cầu các tính chất mạnh hơn so với tính chất mà kỹ thuật băm đều đơn giản cung cấp. Ví dụ, có thể ta muốn các khóa phải “đóng” theo một nghĩa nào đó để cho ra các giá trị băm tách xa nhau. (Tính chất này đặc biệt thỏa đáng khi ta dùng phương pháp thăm dò tuyến tính, được định nghĩa trong phần 2.4. Định địa chỉ mở).
* **Chuyển các khóa dưới dạng các số tự nhiên**

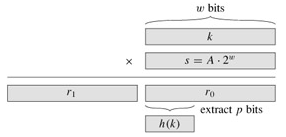
Hầu hết các hàm băm đều mặc nhận không gian của các khóa là tập các số tự nhiên N= {0, 1, 2,…}. Như vậy, nếu các khóa không phải là các số tự nhiên, ta phải có một cách nào đó để chuyển chúng về dưới dạng các số tự nhiên. Ví dụ, một khóa là một chuỗi ký tự có thể được chuyển về dưới dạng một số nguyên được biểu diễn theo cơ số thích hợp. Như vậy, dấu định danh *pt* có thể được chuyển dưới dạng cặp số nguyên thập phân (112, 116), bởi *p* = 112 và *t* = 116 trong bảng mã ASCII; vậy, được diễn tả dưới dạng một số nguyên cơ số -128, *pt* trở thành (112\*128) +116 = 14452. Nói chung trong bất kỳ ứng dụng đã cho nào, ta có thể dễ dàng nghĩ ra một phương pháp đơn giản như vậy để chuyển từng khóa về dưới dạng một số tự nhiên (có thể lớn). Trong nội dung dưới đây, ta mặc nhận các khóa là các số tự nhiên.

## Phương pháp chia

* Trong phương pháp chia để tạo các hàm băm, ta ánh xạ một khóa k vào một trong số m khe bằng cách chia k cho m và lấy số dư. Nghĩa là, hàm băm: h(k) = k mod m.
* Ví dụ, nếu bảng băm có kích cỡ m = 12 và khóa là k = 100, thì h(k) = 4. Vì chỉ yêu cầu một phép chia đơn lẻ nên kỹ thuật băm bằng phép chia chạy khá nhanh.
* Khi dùng phương pháp chia ta thường tránh một số giá trị nhất định của m. Ví dụ, m không được là lũy thừa của 2, bởi nếu m = 2p, thì h(k) chỉ là p bit thứ tự thấp nhất của k. Trừ phi biết trước rằng phép phân phối xác suất trên các khóa khiến tất cả các khuôn mẫu p-bit thứ tự thấp có khả năng như nhau, tốt nhất ta nên để hàm băm tùy thuộc vào tất cả các bit của khóa. Phải tránh các lũy thừa của 10 nếu ứng dụng xử lý các số thập phân dưới dạng các khóa, bởi vậy hàm băm không tùy thuộc vào tất cả các chữ số thập phân của k. Cuối cùng, có thể thấy rằng khi m = 2p – 1 và k là một chuỗi ký tự được chuyển theo cơ số 2p, hai chuỗi đồng nhất ngoại trừ sự hoán vị của hai ký tự kề sẽ băm theo cùng giá trị.
* Các giá trị tốt cho m là các số nguyên tố không quá sát với các lũy thừa chính xác của 2. Ví dụ, giả sử ta muốn phân bổ một bảng băm, có các xung đột được giải quyết bằng dây chuyền, để lưu giữ n = 2000 chuỗi ký tự, ở đó mỗi ký tự có 8 bit. Ta không quan tâm xem xét một số trung bình của 3 thành phần trong một đợt tìm kiếm không thành công, do đó ta phân bổ một bảng băm có kích cỡ m = 701. Sở dĩ con số 701 được chọn bởi vì nó là một số nguyên tố gần α = 2000/3 nhưng không gần bất kỳ lũy thừa nào của 2. Xem mỗi khóa k như một số nguyên, hàm băm sẽ là: h(k) = k mod 701

## Phương pháp nhân

* Phương pháp này sẽ tạo các hàm băm thông qua hai bước. Trước hết, ta nhân khóa *k* với một hằng *A* trong miền giá trị 0 < A < 1 và trích phần phân số của *kA*. Sau đó, ta nhân giá trị này với *m* và lấy sàn của kết quả. Tóm lại, hàm băm là: *h(k) = ⌊m(kA mod 1)⌋*, Ở đó “*kA mod 1*” có nghĩa là phần phân số của *kA*, nghĩa là *kA – ⌊kA⌋*.
* Ưu điểm của phương pháp này là giá trị của *m* là không tới hạn. Ta thường chọn giá trị *m* là một lũy thừa của 2 (m = 2p với một số nguyên p). Bởi vậy ta có thể dễ dàng thực thi hàm trên hầu hết các máy tính như sau: Giả sử kích cỡ từ của máy là w bit và k vừa một từ đơn lẻ. Chúng ta có thể hạn chế giá trị của A là một phần của s/2w, trong đó s là một số nguyên trong khoảng 0 < s < 2w. Tham khảo hình bên dưới, trước tiên ta nhân k với số nguyên w-bit, s = A \* 2w. Kết quả là một giá trị 2w-bit r1 2w + r0, ở đó r1 là từ thứ tự cao của tích và r0 là từ thứ tự thấp của tích. Giá trị băm p-bit muốn có, bao gồm p bit quan trọng nhất của r0.



***Hình 4:*** *Phương pháp nhân của kỹ thuật băm. Phần biểu diễn w-bit của khóa k được nhân với giá trị w-bit s = A\*2w. p bit cao nhất của nửa w-bit thấp của tích lập thành giá trị băm muốn có h(k).*

* Mặc dù phương pháp này làm việc với mọi giá trị của hằng A, song với vài giá trị nó làm việc tốt hơn so với các giá trị khác. Chọn lựa tối ưu tùy thuộc vào các đặc tính của dữ liệu đang được băm. Knuth đề cập chi tiết sự lựa chọn của A và gợi ý rằng:
* A ≈ (√5 -1)/2 = 0.6180339887…Ví dụ, nếu ta có k = 123456, p = 14, m = 214 = 16384, và w = 32. Theo đề xuất của Knuth, A = s/232 = 2654435769/232. Vậy k.s = 32770602229664 = (76300 · 232) + 17612864, và vì vậy r1 = 76300 và r0 = 17612864. 14 bit có nghĩa nhất của r0 sinh ra giá trị của h(k) = 67.

## Kĩ thuật phổ băm

* Trường hợp xấu, nếu chọn các khóa để băm sao cho chọn n khóa mà tất cả đều băm theo cùng khe, cho ra một thời gian truy cập trung bình là *Θ*(n). Mọi hàm băm cố định đều dễ bị tác động bởi trường hợp xấu như trên. Cách hiệu quả duy nhất để cải thiện tình huống đó là chọn hàm băm một cách ngẫu nhiên sao cho độc lập với các khóa thực tế sẽ được lưu trữ. Cách tiếp cận này có tên là *kỹ thuật phổ băm*, cho ra khả năng thực hiện tốt tính theo trung bình, bất chấp việc chọn các loại khóa bất kỳ.
* Ý tưởng chính của kỹ thuật phổ băm đó là lựa chọn hàm băm theo ngẫu nhiên vào thời gian thực hiện từ một lớp các hàm được thiết kế cẩn thận. Như trong trường hợp sắp xếp nhanh, việc ngẫu nhiên hóa bảo đảm không có đầu vào duy nhất nào sẽ luôn gợi đến cách xử lý trường hợp xấu nhất. Do sự ngẫu nhiên hóa, thuật toán có thể xử lý khác nhau trên mỗi lần thực hiện, thậm chí với cùng đầu vào. Quay lại ví dụ về bảng ký hiệu của bộ biên dịch, ta thấy rằng sự lựa chọn các dấu định danh của lập trình viên giờ đây không thể gây ra khả năng thực hiện kỹ thuật băm tồi một cách nhất quán. Khả năng thực hiện tồi chỉ xảy ra nếu bộ biên dịch chọn một hàm băm ngẫu nhiên khiến tập hợp các dấu định danh băm tồi, nhưng xác suất của trường hợp này không lớn và không giống nhau với bất kỳ tập hợp các dấu định danh nào có cùng kích cỡ.
* Cho ℋ là một tập hợp hữu hạn các hàm băm ánh xạ một không gian *U* các khóa đã cho vào miền giá trị {0, 1, …, m-1}. Một tập hợp như vậy được gọi là *phổ băm* nếu với mỗi cặp khóa riêng biệt *k*, *l ∈ U*, số lượng hàm băm *h ∈ ℋ* qua đó *h(k)* = *h(l)* chính xác là *|ℋ|/m*. Nói cách khác, với một hàm băm được chọn ngẫu nhiên từ *ℋ*, cơ hội của một xung đột giữa *k* và *l* khi   
  *k≠ l* sẽ không lớn hơn *1/m* cơ hội của một xung đột nếu *h(l)* và *h(k)* được chọn ngẫu nhiên từ tập hợp {0, 1, …, m-1}.Định lý dưới đây cho thấy một lớp toàn thể gồm các hàm băm cho ta cách xử lý trường hợp trung bình tốt.
* *Định lý 3*

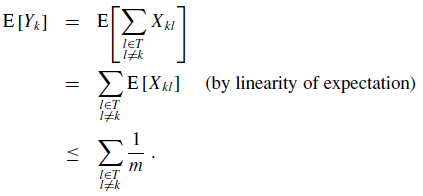
Nếu *h* được chọn từ một tập hợp các hàm băm được gọi là lớp phổ băm và được dùng để băm *n* khóa vào một bảng *T* có kích cỡ *m*, sử dụng phương pháp dây chuyền để giải quyết xung đột. Nếu khóa *k* không có trong bảng, thì độ dài kỳ vọng *E[nh(k)]* của danh sách khóa *k* tối đa là *α*. Nếu khóa *k* có trong bảng, thì độ dài kỳ vọng *E[nh(k)]* của danh sách khóa *k* tối đa là *1+ α*.

* Chứng minh

Ta lưu ý rằng những kỳ vọng hơn là sự lựa chọn của hàm băm, và không phụ thuộc bất kỳ giả định về việc phân phối của các khóa. Mỗi cặp k, l các khóa riêng biệt, xác định biến ngẫu nhiên Xkl = I{h(k) = h(l)}. Vì theo định nghĩa, một cặp khóa với xác suất xung đột nhiều nhất là 1/m, chúng ta có Pr (h (k) = h (l)) ≤ 1 / m, theo Lemma (chương trước) E[Xkl] ≤ 1 / m.Tiếp theo, chúng ta xác định đối với mỗi khóa k, biến ngẫu nhiên Yk số lượng các khóa mà các khóa được băm vào cùng khe với khóa k:



* Vậy chúng ta có



* Phần còn lại của việc chứng minh, ta xác định k có trong bảng T hay không.
  + Nếu k ∉ T, thì nh(k) = Yk và |{l : l ∈ T và l ≠ k}| = n. Vậy E[nh(k)] = E[Yk] ≤ n/m = α.
  + Nếu k ∈ T , vì khóa k xuất hiện trong T[h(k)] và Yk không bao gồm k, ta có nh(k) = Yk + 1 và|{l : l ∈ T và l ≠ k}| = n - 1. Vậy E[nh(k)]= E[Yk] + 1 ≤ (n - 1)/m + 1 = 1 + α - 1/m < 1 + α.
* Hệ quả : Sử dụng phổ băm và giải quyết các xung đột trong bảng với m khe, phải mất thời gian dự kiến Θ(n) để xử lý tuần tự n phép chèn, tìm kiếm và xóa trong O(m) phép chèn.
* Chứng minh
  + Số các phép chèn là O(m), ta có n = O(m) và do đó α = O(1). Phép chèn và phép xóa là các hoạt động liên tục và mất thời gian. Theo định lý 3, thời gian dự kiến cho phép tìm kiếm là O(1). Bởi tính chất tuyến tính của kỳ vọng, vì vậy thời gian dự kiến cho toàn bộ hoạt động là O(n).
* ***Thiết kế một lớp các hàm phổ băm***
* Khá dễ dàng để thiết kế một lớp cho các hàm phổ băm. Đầu tiên chọn một số nguyên tố p đủ lớn để mọi khóa k có giá trị trong khoảng từ 0 đến p-1. Zp là tập hợp (0, 1, …, p-1) và Z\*p là tập hợp {1, 2, …, p-1}. Vì p là số nguyên tố, chúng ta có thể giải quyết phương trình module p với những phương pháp được đưa ra trong chương Số - Lý thuyết thuật toán. Bởi vì số lượng của các khóa lớn hơn số lượng các khe của bảng băm, tức là p > m.
* Bây giờ chúng ta xác định hàm băm ha,b với mọi b ∈ Zp sử dụng phép biến đổi tuyến tính rút gọn phân số của phép chia modulo p và modulo m: ha,b(k) = ((ak + b) mod p) mod m. (2)
* Ví dụ: Với p = 17 và m = 6. Ta có H3,4(8) = 5. Các hàm băm tương tự có dạng: ℋp,m = { ha,b:a ∈ Z\*p và b ∈ Zp } (3)
* Mỗi hàm băm ℋp,m ánh xạ Zp đến Zm. Có p -1 lựa chọn a và p lựa chọn b, vì vậy có p(p-1) hàm băm trong lớp ℋp,m.
* Định lý 4: Lớp các hàm băm ℋp,m được chỉ ra trong *(2)* và *(3)* là các lớp phổ băm.
* ***Chứng minh***
  + Xét hai khóa riêng biệt k và l trong tập Zp, k ≠ l. Xác định hàm băm ha,b:

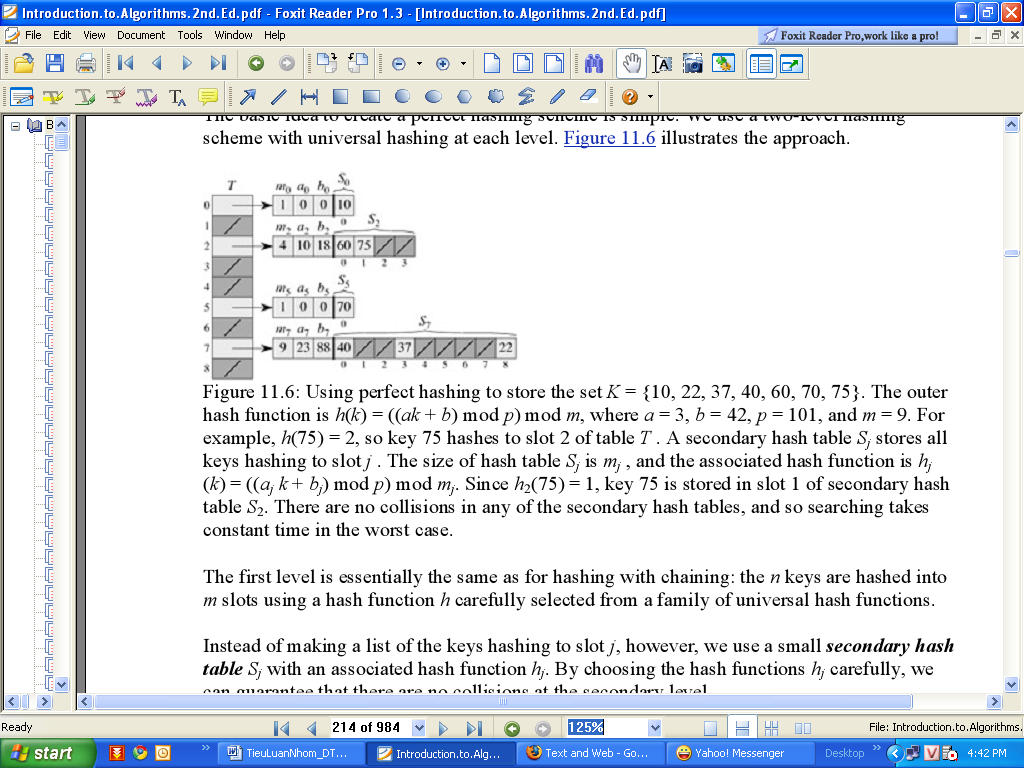
r = (ak + b) mod p

s = (al + b) mod p

* Trước tiên, cần chú ý r ≠ s. Vì sao như vậy?
* r – s ≡ a(k - l) (mod p).
* Ta có r ≠ s vì p là số nguyên tố, a mod p ≠ 0 và (k – l) mod p ≠ 0. Vậy   
  a(k-l) mod p ≠ 0. Vì vậy trong quá trình xác định bắt kỳ hàm băm ha,b trong lớp ℋp,m, các giá trị phân biệt k và l ánh xạ đến các giá trị các giá trị phân biệt trong phép chia lấy dư của r và s cho p; không xuất hiện xung đột tại “mức mod p”. Hơn nữa mỗi p(p -1) lựa chọn của cặp (a,b) với a ≠ 0 tương ứng với các kết quả khác nhau của cặp (r,s) với r ≠ s.
* a = ((r - s)((k - l)-1 mod p)) mod p
* b = (r - ak) mod p
* Phép ((k - l)-1 mod p) là phép nghịch đảo của (k-1) sau đó mod p. Vì p(p-1) cặp (r,s) với r ≠ s có mối quan hệ một – một với cặp (a,b) với a ≠ 0. Vì vậy, đối với bất kỳ cặp cho đầu vào k và l, nếu chúng ta chọn (a, b) thống nhất ngẫu nhiên từ Z\*p×Zp, các cặp kết quả (r, s) là bình đẳng với khả năng nhận được từ các cặp giá trị khác biệt modulo p.
* Khả năng xung đột của các khóa k và l bằng với khả năng r ≡ s (mod m) khi mà r và s là các giá trị ngẫu nhiên từ phép modulo p. Đối với một giá trị nhất định của r, có p - 1 lựa chọn giá trị cho s, số lượng các giá trị s sao cho s ≠ r và s ≡ r (mod m) nhiều nhất là: ⌈p/m⌉ - 1 ≤ ((p + m - 1)/m) - 1 = (p - 1)/m.
* Xác xuất s xung đột với r khi ((p − 1)/m)/(p − 1) = 1/m. Vì vậy với mọi cặp phân biệt k và l ∈ Zp,Pr{ha,b(k) = ha,b(l)} ≤ 1/m,
* Do đó lớp các hàm băm ℋ*p,m* thực sự là lớp phổ băm.

## Kĩ thuật băm hoàn hảo

* Mặc dù băm thường được sử dụng cho hiệu quả thực hiện tốt, băm có thể được sử dụng để có được thực hiện tốt trong *trường hợp xấu nhất* khi thực hiện thiết lập các khóa là *tĩnh*: một khi các khóa được lưu trữ trong bảng, tập các khóa không bao giờ thay đổi. Một số ứng dụng tự nhiên có tập các khóa tĩnh: xem xét việc thiết lập các từ trong một ngôn ngữ lập trình, hoặc thiết lập các tập tin trên đĩa CD-ROM. Chúng tôi gọi một kỹ thuật là *băm hoàn hảo* nếu trường hợp xấu nhất truy cập bộ nhớ cần thiết để thực hiện việc tìm kiếm là *O*(1).
* Ý tưởng cơ bản để tạo ra lược đồ băm hoàn hảo khá đơn giản. Chúng tôi sử dụng lược đồ băm hai cấp với băm tổng thể ở mỗi cấp. Hình 6 minh họa phương pháp tiếp cận.



***Hình 6****: Sử dụng băm hoàn hảo để lưu trữ tập K = {10, 22, 37, 40, 60, 70, 75}.*

* Hàm băm bên ngoài là h(k) = ((ak + b) mod p) mod m, ở đây a = 3, b = 42, p = 101, và m = 9. Ví dụ, h(75) = 2, do đó khóa 75 băm vào khe 2 của bảng T. Một bảng băm phụ Sj lưu trữ tất cả các khóa băm vào khe j. Kích thước của bảng băm Sj là m­j, và hàm băm liên quan là hj(k) = ((ajk + bj) mod b) mod mj. Vì h2(75) = 1, khóa 75 là đuợc lưu trữ ở khe 1 của bảng băm phụ S2. Không có những xung đột bất kì ở bảng băm phụ, và vì thế thời gian tìm kiếm trong trường hợp xấu nhất là một hằng số.
* Mức cơ bản đầu tiên là giống như cho băm với loạt: *n* khóa được băm vào *m* khe sử dụng hàm băm *h* được lựa chọn một cách cẩn thận từ họ của hàm băm tổng thể.
* Tuy nhiên, thay vì cho một danh sách các khóa băm vào khe *j*, chúng tôi sử dụng một ***bảng băm phụ*** nhỏ *Sj* liên kết với hàm băm *hj*­. Bằng cách chọn hàm băm *hj* một cách cẩn thận chúng tôi có thể đảm bảo rằng không có xung đột ở mức 2.
* Tuy nhiên, để đảm bảo rằng là không có xung đột ở mức 2, chúng ta sẽ cần để kích thước *mj* của bảng băm *Sj* là bình phương của số *nj* của các khóa băm vào khe *j*. Trong khi đó có một sự phụ thuộc vào phương trình bậc 2 của *mj* vào *nj*. Dường như xảy ra yêu cầu lưu trữ tổng thể là vượt quá giới hạn, chúng ta sẽ chỉ ra bằng cách chọn mức đầu tiên của hàm băm là tốt, tổng không gian dự kiến được sử dụng vẫn là *O*(n).
* Chúng ta sử dụng các hàm băm được chọn từ tổng thể các lớp của hàm băm.
* Mức đầu tiên của hàm băm là được chọn từ lớp , ở đây, *p* là một khóa chính mà lớn hơn bất kì giá trị khóa nào. Những khóa băm đến khe *j* là được băm lại ở bảng băm phụ *Sj* với kích thước *mj* sử dụng hàng băm *hj* được chọn từ lớp 
* Chúng ta sẽ tiến hành trong 2 bước. Đầu tiên chúng ta sẽ xác định cách để đảm bảo rằng bảng phụ không có xung đột. Tiếp theo, chúng ta sẽ chỉ ra rằng số lượng tổng thể của bộ nhớ được sử dụng mong đợi – cho bảng băm chính và tất cả các bảng băm phụ là *O*(n).
* **Định lý 7**

Nếu chúng ta lưu trữ *n* khóa trong một bảng băm có kích thước *m = n2* sử dụng hàm băm *h* được chọn ngẫu nhiên từ lớp tổng của các hàm băm, thì xác suất có được bất kì sự xung đột là nhỏ hơn 1/2 .

* ***Chứng minh***

Có cặp khóa  mà xảy ra xung đột; mỗi cặp xung đột với xác suất là *1/m*. Nếu *h* là được chọn ngẫu nhiên từ họ tổng thể *H* của hàm băm. Cho *X* là một biến ngẫu nhiên số các xung đột. Khi *m = n2*, số xung đột được mong đợi đó là:

  .

Áp dụng bất đẳng thức Markov’s (C.29), Pr{X >= t} <= E[X]/t, với t=1, điều phải chứng minh.

Trong tình huống được mô tả ở định lý 7, ở đây *m = n2*, cho rằng một hàm băm *h* là được chọn ngẫu nhiên từ *H* là nhiều khả năng hơn không phải không có xung đột. Cho tập *K* gồm *n* khóa được băm (lưu ý rằng *K* là tĩnh), do vậy dễ nhận ra một xung đột tự do ở hàm băm *H* với một vài thử nghiệm ngẫu nhiên.

Tuy nhiên, khi *n* là lớn thì một bảng băm có kích thước *m = n2*là quá nhiều. Vì vậy chúng ta áp dụng tiếp cận băm 2 cấp, và sử dụng cách tiếp cận của định lý 7, chỉ để băm vào mỗi phần tử bên trong các khe. Một hàm băm bên ngoài, hoặc là mức đầu tiên, hàm băm *h* là được sử dụng để băm vào các khe *m = n*. Sau đó nếu khóa *nj*băm vào khe *j*, bảng băm phụ *Sj* có kích thước *mj = n2j* được sử dụng để mà cung cấp xung đột tự do với thời gian là hằng số.

Bây giờ chúng ta lần lượt đến vấn đề bảo đảm bộ nhớ tổng thể được sử dụng là *O*(n). Khi kích thước *mj* với chỉ số *j* ở bảng băm phụ tuân theo phương trình bậc 2 với số khóa nj được lưu trữ. Có một nguy cơ là số tổng thể được lưu trữ có thể là quá nhiều.

Nếu mức đầu tiên của bảng có kích thước là *m = n*, thì số lượng bộ được sử dụng là *O*(n) cho bảng băm chính, cho lưu trữ của kích thích mj của bảng băm phụ và cho lưu trữ các tham số: ajbj xác định hàm băm phụ hj rút ra từ lớp (trừ khi nj = 1 và a = b = 0). Định lý sau đây và hệ quả cung cấp một phạm vi được mong đợi kết hợp các kích thước của tất cả các bảng băm phụ. Một hệ quả thứ 2 là kết hợp phạm vi xác suất kích thước của tất cả các bảng băm phụ là siêu tuyến tính.

* **Định lí 8**

Nếu chúng ta lưu trữ *n* khóa trong bảng băm có kích thước *m = n*, sử dụng một hàm băm h được chọn ngẫu nhiên từ lớp tổng thể của các hàm băm thì:



Ở đây *nj* là số các khóa băm vào khe *j*.

* ***Chứng minh***

Chúng ta bắt đầu với định nghĩa, cho bất kì số nguyên không âm a

. (5)

Ta có:

 (theo phương trình (5))

 (theo tính chất của kỳ vọng)



 (với n không là biến ngẫu nhiên)

Để tính tổng , chúng ta nhận thấy rằng nó là tổng số các xung đột bởi các thuộc tính của băm tổng thể, giá trị được mong đợi của tổng này là:

 ,

Khi m = n. Do đó:

 .

* **Hệ quả 1**

Nếu chúng ta lưu *n* khóa ở bảng băm có kích thước *m = n* sử dụng hàm băm *h* được chọn ngẫu nhiên từ lớp tổng thể các hàm băm và chúng ta đặt kích thước của mỗi bảng băm phụ là  với j = 0, 1, …, m-1, thì số lượng lưu trữ mong đợi đối với tất cả bảng băm phụ trong lược đồ băm hoàn hảo là nhỏ hơn 2n.

* ***Chứng minh***

Khi  với j = 0, 1, …, m-1, Định lý 8 cho,  (6), điều phải chứng minh.

* **Hệ quả 2**

Nếu lưu trữ *n* khóa vào bảng băm kích thước *m = n* sử dụng hàm băm được chọn ngẫu nhiên từ lớp tổng thể các hàm băm và chúng ta giả thiết kích thước của mỗi bảng băm phụ là  với *j = 0, 1, …, m-1*, thì xác suất mà tổng số lưu trữ được sử dụng cho bảng băm phụ vượt quá 4n là nhỏ hơn 1/2.

* ***Chứng minh***

Ta áp dụng bất đẳng thức Markov’s một lần nữa, Pr{Xt}  E[X] /t, bất đẳng thức lúc này (6), với  và *t = 4n*:



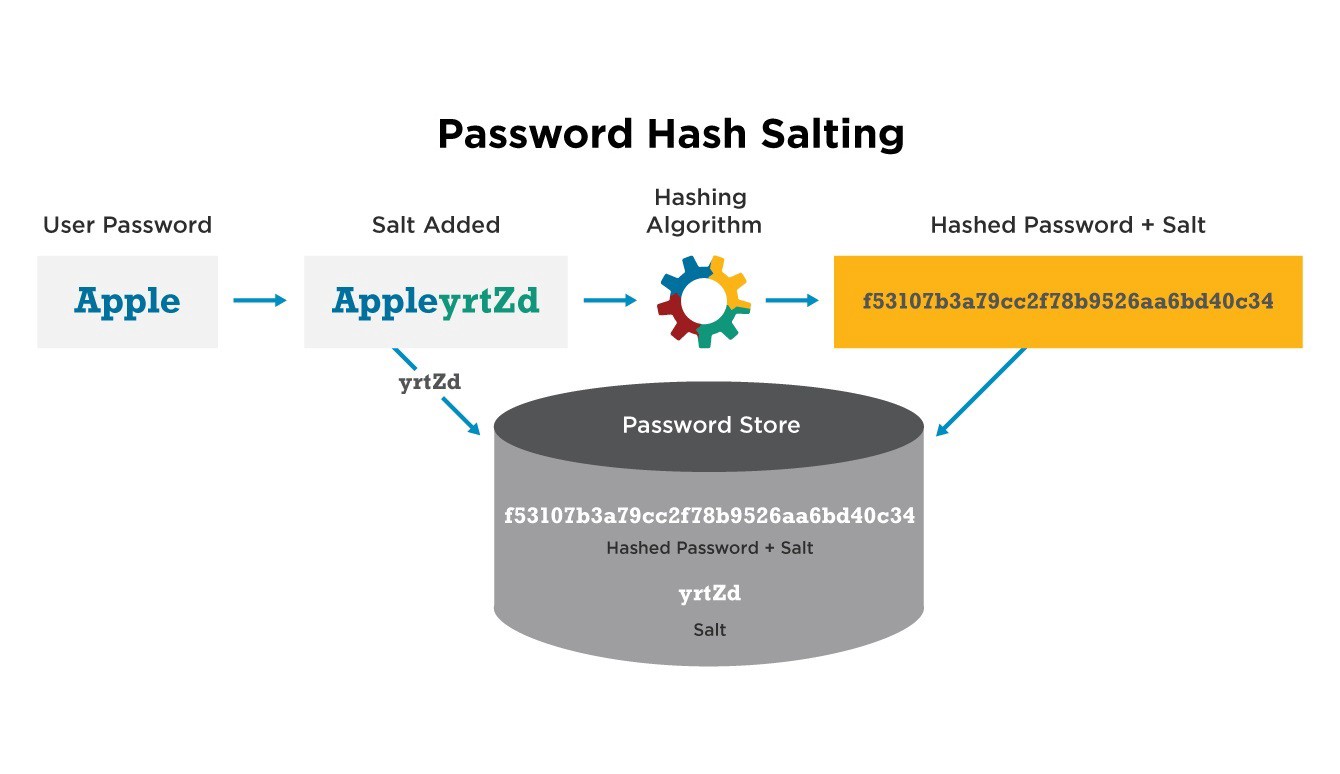


Từ hệ quả 2, chúng ta nhận thấy rằng việc kiểm tra một số hàm băm được chọn ngẫu nhiên từ họ tổng thể sẽ mang lại một cách nhanh chóng dung lượng sử dụng không gian nhớ hợp lý.

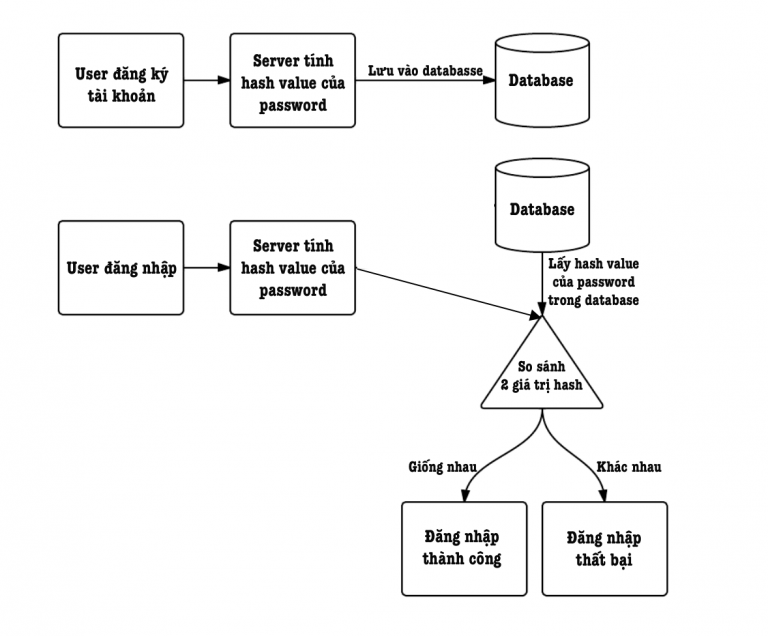
# CHƯƠNG II: ỨNG DỤNG BẢNG BĂM

## CHE DẤU DỮ LIỆU GỐC

* Hash được sử dụng cho mật mã vì nó che dấu dữ liệu gốc bởi một giá trị khác. Vì sử dụng trong mật mã nên một "**hàm băm tốt**" là một hàm băm không thể đảo ngược. Dưới đây là ví dụ của việc sử dụng hash trong xác minh mật khẩu.
* Khi lưu password trong cơ sở dữ liệu (database) người dùng thường dùng kỹ thuật “password hashing” chứ không lưu trực tiếp password. Vì nếu database bị xâm phạm (bởi các lực lượng bên ngoài hoặc bên trong, bên trong ở đây có thể là chính người quản trị cơ sở dữ liệu, bên ngoài ở đây có thể là các hacker, kẻ phá hoại,…) thì tất cả các thông tin về user trong đó có password sẽ bị lộ.
* Thay về lưu password dưới dạng plaintext (dạng text bình thường mà con người có thể đọc hiểu được) thì password sẽ được đưa vào một hash function, hash value sinh ra từ hash function sẽ được dùng để lưu và database đại diện cho password. Dưới đây là sơ đồ của mô hình này



* Cụ thể, nếu đăng ký tài khoản trên trang web với username là “username” và password là “12345678“, webserver sử dụng thuật toán hash để tính hash value của password và lưu xuống database. Giả sử, hash value của “12345678” là “25d55ad283aa400af464c76d713c07ad”

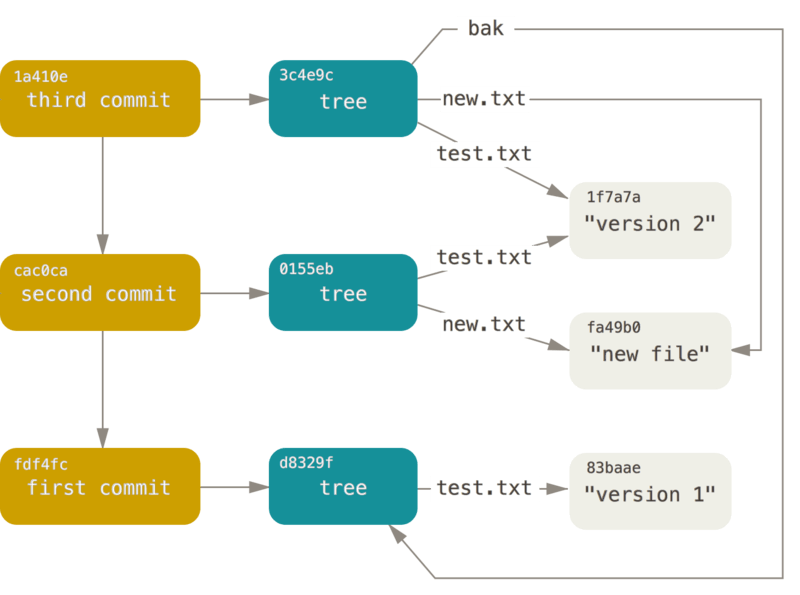


* Lần sau, khi đăng nhập nếu nhập “12345678” trong ô password thì webserver sẽ lấy hash value của dữ liệu vừa nhập vào và so sánh với giá trị “25d55ad283aa400af464c76d713c07ad“ lưu trong database. Trường hợp hai giá trị hash bằng nhau thì đã đăng nhập thành công.
* Ngược lại, nếu nhập “123456789” thì có giá trị hash nhận được khi này sẽ là “25f9e794323b453885f5181f1b624d0b”.

## HASHING TRONG ĐỊNH DANH DỮ LIỆU

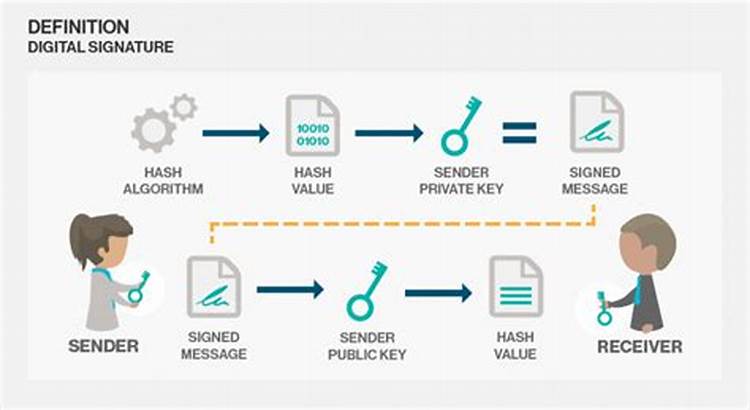
#### **Định danh tệp hoặc dữ liệu**

Giá trị băm cũng có thể được sử dụng như một phương tiện để định danh tập tin một cách đáng tin cậy. Một số hệ thống quản lý mã nguồn, như Git, Mercurial hay Monotone, sử dụng giá trị sha1sum của nội dung tệp, cây thư mục, thông tin thư mục gốc, v.v. để định danh chúng. Những giá trị này còn được sử dụng để xác định các tệp trên các mạng chia sẻ ngang hàng nhằm cung cấp đầy đủ thông tin để định vị nguồn gốc của tệp, xác minh nội dung tệp tải xuống. Giá trị ứng dụng của chúng còn được mở rộng ra khi áp dụng các cấu trúc dữ liệu bổ sung như danh sách băm hoặc cây băm.



#### **Tạo và xác nhận giá trị tổng kiếm hoặc chữ ký số**

***Checksum*** là giá trị tổng kiểm, được sử dụng với mục đích xác nhận tính toàn vẹn của file. Checksum là giá trị nhỏ được tạo dựa trên các bit trong một file hoặc khối dữ liệu như image đĩa. Khi chức năng checksum chạy trên một bản sao của file (chẳng hạn như file được tải từ Internet), nó sẽ tạo ra giá trị băm giống như file gốc. Nếu file không tạo cùng một checksum, có nghĩa là file đã bị thay đổi.Ngoài ra, cũng giống như checksum, hash cũng được sử dụng trong tạo và xác nhận chữ ký số an toàn và hiệu quả.



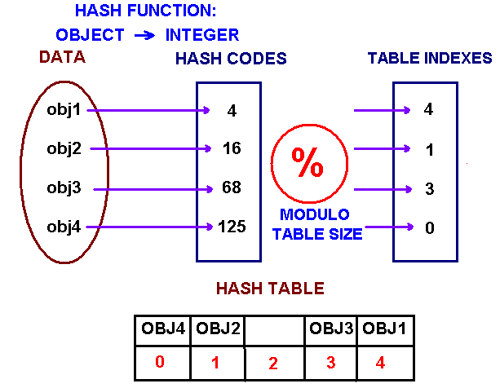
## ****NÉN DỮ LIỆU****

Vì tính chất “tạo ra dữ liệu với kích thước cố định từ một đầu vào bất kỳ” nên việc góp sức hash rõ ràng là một trong các phương thức để giảm kích thước file. Cụ thể, bằng cách hash đầu vào thành giá trị nhỏ, dữ liệu có thể được nén thành các phần nhỏ hơn. Tuy nhiên, do hash còn có một tính chất nữa đó là “tính một chiều” nên khi giải nén ta không thể đảo ngược để có được dữ liệu gốc nhưng có thể tái tạo lại dữ liệu gần bằng dữ liệu gốc.

## ****ỨNG DỤNG TRONG LẬP CHỈ MỤC VÀ TRUY XUẤT DỮ LIỆU****

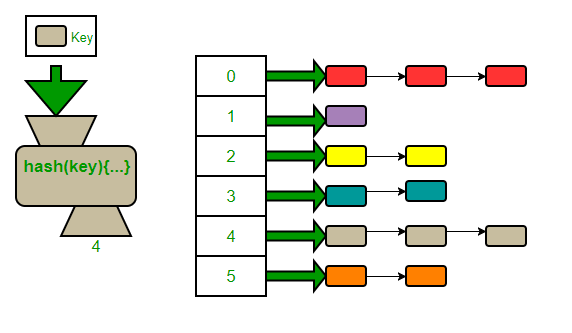
Một trong những ví dụ cụ thể đó là [**hashtable**](https://codelearn.io/sharing/hash-table-manh-nhu-the-nao-trong-policy-based-data-structures). Hashtable được hỗ trợ trong rất nhiều các ngôn ngữ, là một trong những nền tảng về thuật toán và cấu trúc dữ liệu.

Hash table là một cấu trúc dữ liệu dùng để lưu theo các cặp key-value, nó dùng hash function để tính toán ra một index (nơi lưu trữ một bucket các giá trị) rồi từ đó sẽ retrieve ra value.

**

Giả sử có các object với data bất kỳ. Điều đầu tiên, khi đưa dữ liệu qua hash function thì lợi ích chúng ta có được là dữ liệu đầu ra được đồng bộ (đó là hash codes). Tuy nhiên, kích thước bảng vẫn còn lớn nên ta sẽ sử dụng kỹ thuật “Modulo table size” để giảm kích thước xuống và có được một hash table.

Một trong những vấn đề của hash đó là “collision” (trùng vị trí index). Điều này sẽ xảy ra nếu thuật toán hash không được tốt. Và thực tế, hầu như không có thuật toán hash nào thực sự hoàn hảo để sinh ra unique key nếu lưu trữ một lượng lớn dữ liệu. Giải pháp cho vấn đề này là dùng linked list để lưu trữ thêm một tầng nữa các phần tử cho index đó.

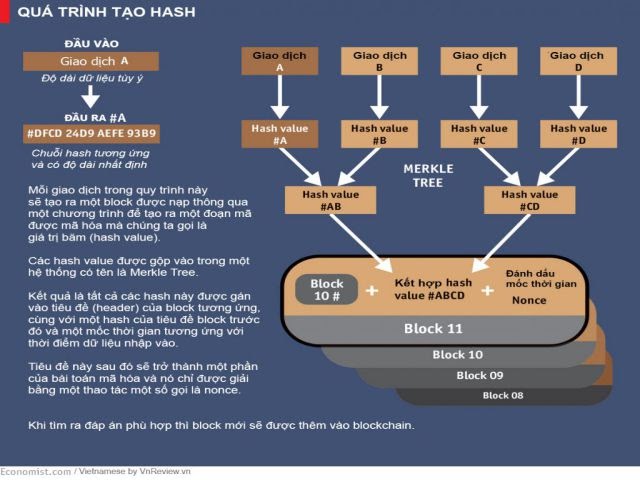


Có thể coi việc sinh hash là để tạo keyHash và lưu vào trong mảng giá trị nơi chứa một linked list có phần tử chứa key thật, sau khi truy xuất tới đó chúng ta sẽ dùng key thật để retrieve giá trị.

Như vậy, việc sử dụng hashtable sẽ giúp tạo một chỉ mục nhỏ hơn nhiều so với dữ liệu gốc. Nhờ đó việc tìm kiếm và truy cập dữ liệu sẽ hiệu quả hơn.

### **ỨNG DỤNG TRONG BẰNG CHỨNG CÔNG VIỆC (PROOF OF WORK)**

* **Proof of work** ̣(PoW) là một biện pháp kinh tế để ngăn chặn các cuộc tấn công từ chối dịch vụ và các hành vi lạm dụng dịch vụ khác như spam bằng cách yêu cầu người dùng dịch vụ thực hiện một số công việc nhất định, thường đòi hỏi nhiều thời gian xử lý. Bằng chứng công việc cần đảm bảo tính bất đối xứng tức là: công việc phải có độ khó vừa phải (nhưng khả thi) về phía người dùng nhưng dễ kiểm chứng đối với nhà cung cấp dịch vụ.
* PoW cũng là thuật toán đồng thuận đầu tiên được tạo ra trong mạng Blockchain. Được sử dụng để xác nhận giao dịch và sản xuất các block mới trong chuỗi.
* PoW yêu cầu những người sở hữu các máy tính trong mạng phải giải một bài toán phức tạp để có thể thêm một block (khối) vào chuỗi. Tuy nhiên, không dễ để có câu trả lời cho vấn đề toán học. Các bạn có thể hình dung mô hình hoạt động qua hình sau:



**TÀI LIỆU THAM KHẢO :**

<https://nguyenvanhieu.vn/bang-bam-hash-tables/>

<https://codelearn.io/sharing/5-ung-dung-thu-vi-cua-hash?fbclid=IwAR2eOliCXjFaFbCMx5W18qrRurBsDjkhQrvn8X-Es7DdaCX1xoJpd8iTxdg>

<https://vi.wikipedia.org/wiki/B%E1%BA%A3ng_b%C4%83>m

<https://tek4.vn/khoa-hoc/cau-truc-du-lieu-va-giai-thuat/cau-truc-du-lieu-bang-bam-va-ung-dung-cua-bang-bam>

<https://cryptoviet.com/hash-la-gi>

<https://www.geeksforgeeks.org/what-are-hash-functions-and-how-to-choose-a-good-hash-function/>

<https://www.tutorialspoint.com/cryptography/cryptography_hash_functions.htm>