VIETNAM NATIONAL UNIVERSITY - HO CHI MINH

UNIVERSITY OF SCIENCE

Universal & Perfect Hashing

Techincal Report

CSC10004 - Data Structure And Algorithm

Students:

24120059 - Trần Kim Hữu

24120041 - Phạm Võ Đức

24120006 - Đào Thanh Phong

24120069 - Trần Hoài Bảo Khang

Supervisors:

Lect. Lê Nhựt Nam

Ngày 10 tháng 6 năm 2025

MỤC LỤC

DANH MỤC CÁC BẢNG

DANH MỤC CÁC HÌNH VẼ, ĐỒ THỊ

LỜI CẢM ƠN

Trong thế giới kỹ thuật số ngày nay, khi dữ liệu tăng lên theo cấp số nhân mỗi giây, việc tìm kiếm những phương pháp hiệu quả để lưu trữ và truy xuất thông tin trở nên vô cùng quan trọng. Các hệ thống hiện đại, từ trình biên dịch, công cụ tìm kiếm, cơ sở dữ liệu, cho đến các hệ thống nhúng và phần mềm thời gian thực đều đòi hỏi khả năng truy cập dữ liệu cực nhanh và ổn định. Trong bối cảnh đó, các kỹ thuật băm – đặc biệt là Universal Hashing và Perfect Hashing – đóng vai trò then chốt trong việc xây dựng các cấu trúc dữ liệu có hiệu suất cao, đảm bảo truy cập trong thời gian hằng định, ngay cả trong trường hợp xấu nhất.

Đề tài "Universal and Perfect Hashing" không chỉ là một nghiên cứu học thuật đơn thuần mà còn là một hành trình khám phá sâu sắc về cách tổ chức dữ liệu tối ưu và kỹ thuật thiết kế hàm băm ngẫu nhiên đảm bảo độ an toàn và hiệu suất. Việc tìm hiểu các lý thuyết nền tảng như hàm băm phổ quát (universal hashing), phân tích xác suất, đánh giá độ phức tạp về thời gian và không gian đã giúp chúng em hiểu rõ hơn về mối liên hệ giữa lý thuyết toán học và ứng dụng trong tin học.

Chúng em xin bày tỏ lòng biết ơn sâu sắc nhất đến thầy Lê Nhựt Nam – người đã tận tình hướng dẫn, truyền đạt kiến thức và tạo điều kiện, hỗ trợ chúng em trong suốt quá trình học tập và nghiên cứu. Đề tài này là cơ hội quý báu giúp chúng em kết nối kiến thức lý thuyết với ứng dụng thực tế, phát triển kỹ năng phân tích, tư duy thuật toán và khả năng trình bày khoa học.

Chúng em xin chân thành cảm ơn.

TÓM TẮT

Trong khoa học máy tính, việc thiết kế cấu trúc dữ liệu hiệu quả cho phép tăng cường hiệu suất truy xuất thông tin trong các hệ thống hiện đại. Bảng băm (Hash Table) là một cấu trúc dữ liệu được sử dụng phổ biến nhờ khả năng thực hiện các phép toán tìm kiếm, chèn và xoá với thời gian trung bình $O(1)$. Tuy nhiên, các bảng băm truyền thống phụ thuộc mạnh vào chất lượng của hàm băm và gặp khó khăn trong trường hợp xấu nhất, khi xảy ra nhiều xung đột khóa. Điều này đặc biệt nghiêm trọng trong các hệ thống yêu cầu hiệu năng cao và ổn định như trình biên dịch, hệ thống mạng hoặc cơ sở dữ liệu lớn.

Đề tài "Universal and Perfect Hashing" nghiên cứu kỹ thuật băm hoàn hảo hai cấp (two-level Perfect Hashing) áp dụng hàm băm phổ quát (Universal Hashing) nhằm đảm bảo truy vấn dữ liệu luôn đạt thời gian hằng định $O(1)$, ngay cả trong trường hợp xấu nhất. Phương pháp được xây dựng trên nền lý thuyết xác suất và thiết kế hàm băm ngẫu nhiên sao cho khả năng xảy ra xung đột là cực kỳ thấp.

Cấu trúc băm được triển khai như sau:

- **Cấp 1:** Sử dụng một hàm băm từ họ Universal Hashing để phân phối n khóa vào m bucket, trong đó m là số nguyên tố nhỏ nhất lớn hơn hoặc bằng n.
- **Cấp 2:** Với mỗi bucket chứa từ hai phần tử trở lên, một bảng băm phụ có kích thước $4n_i^2$ được khởi tạo và một hàm băm riêng được chọn để đảm bảo không có xung đột trong bucket đó.

Tập dữ liệu được sử dụng là các file văn bản chứa từ 45.000 đến 370.000 dòng, mỗi dòng là một khóa. Sau khi xây dựng bảng băm, mỗi khóa được truy vấn lặp lại 10^6 lần để tính thời gian trung bình.

Kết quả thực nghiệm cho thấy thời gian truy xuất trung bình đạt mức ổn định, dao động trong khoảng 120–170 picosecond cho mỗi khóa. Thời gian xây dựng bảng băm dao động từ 10ms đến 142ms tùy theo kích thước tập dữ liệu. Số lần thử lại để chọn được hàm băm cấp 2 phù hợp trung bình nhỏ hơn 2 lần cho mỗi bucket.

Từ kết quả trên, có thể khẳng định kỹ thuật **Perfect Hashing** cho hiệu suất truy xuất cao, ổn

định và phù hợp cho các hệ thống có tập khóa tĩnh như từ điển, bảng ký hiệu, hệ thống chỉ đọc hoặc định tuyến mạng. Tuy nhiên, phương pháp này không thích hợp với các bài toán cần cập nhật dữ liệu thường xuyên, do chi phí xây dựng lại bảng băm cao.
Để đánh giá hiệu năng của kỹ thuật băm hai cấp, chương trình được kiểm thử trên ba bộ dữ liệu đầu vào với quy mô từ vài nghìn đến hàng trăm nghìn dòng (tương ứng từ 5% đến 100% tổng tập dữ liệu). Các cấp băm được thay đổi từ bậc 1 đến bậc 6 để phân tích tác động của kích thước bảng phụ. Kết quả đo thời gian tra cứu trung bình (10^6 lần truy xuất trên 5 từ khóa) cho thấy:

- Với bảng phụ cấp 1 (kích thước $\approx n$), hiệu năng giảm mạnh khi quy mô dữ liệu lớn, do xảy ra nhiều xung đột.
- Từ cấp 2 trở lên, thời gian tra cứu cải thiện đáng kể và trở nên ổn định ở mức 120–130 picosecond bất kể kích thước tập dữ liệu.
- Với bảng phụ kích thước bậc 4 ($\approx 4n^2$), hiệu năng đạt mức cao nhất và ổn định nhất. Thời gian xây dựng bảng dao động từ 10ms đến khoảng 140ms, ngay cả khi số lượng dòng lên đến 72.000.

Nhìn chung, cấp độ băm ảnh hưởng trực tiếp đến khả năng loại bỏ xung đột và tối ưu hiệu suất tra cứu. Tuy nhiên, việc chọn bậc quá cao cũng làm tăng chi phí bộ nhớ. Vì vậy, trong thực tiễn, việc lựa chọn bậc phù hợp (thường từ 2 đến 4) sẽ cân bằng tốt giữa hiệu suất và tài nguyên sử dụng.

Tổng kết, đề tài đã triển khai thành công cấu trúc băm hoàn hảo hai cấp trên nền Universal Hashing, từ lý thuyết cho đến ứng dụng thực nghiệm. Đây là tiền đề quan trọng cho các nghiên cứu mở rộng như minimal perfect hashing, dynamic perfect hashing hoặc áp dụng trong các cơ sở dữ liệu đặc thù.

CHƯƠNG I

GIỚI THIỆU

1. Bối cảnh

Trong khoa học máy tính, các cấu trúc dữ liệu hiệu quả là nền tảng cho mọi hệ thống phần mềm hiện đại, từ cơ sở dữ liệu, trình biên dịch, đến các hệ thống phân tán và trí tuệ nhân tạo. Trong đó, **hash table** (bảng băm) là một trong những cấu trúc phổ biến nhất nhờ khả năng *tra cứu nhanh (lookup)*, *chèn (insert)* và *xoá (delete)* gần như tức thì trong thực tế.

Tuy nhiên, hầu hết các bảng băm truyền thống đều dựa vào *giả định trung bình*, tức là thời gian truy cập $O(1)$ chỉ xảy ra khi hàm băm phân phối đều và có ít xung đột. Trong *trường hợp xấu (worst-case)*, có thể xảy ra nhiều xung đột, dẫn đến thời gian truy xuất tăng đáng kể, ảnh hưởng nghiêm trọng đến hiệu năng.

Để khắc phục điều này, các nhà khoa học đã phát triển kỹ thuật **Perfect Hashing** – một phương pháp xây dựng bảng băm mà **không xảy ra xung đột**, cho phép tra cứu **luôn luôn trong thời gian** $O(1)$, ngay cả trong trường hợp xấu nhất.

2. Lịch sử

Vấn đề xây dựng **Perfect Hashing** cho tập dữ liệu tĩnh đã được đặt ra từ sớm, trong các hệ thống nơi dữ liệu không thay đổi sau khi khởi tạo, ví dụ như từ điển ngôn ngữ, bảng ký hiệu trình biên dịch, hay bảng tra cứu hằng số.

Một bước ngoặt quan trọng xảy ra vào năm 1984, khi ba nhà khoa học Michael Fredman, János Komlós và Endre Szemerédi công bố công trình nổi tiếng: *"Storing a Sparse Table with O(1) Worst Case Access Time"* [2]

Công trình của họ giới thiệu khái niệm **Perfect Hashing**, trong đó họ chứng minh rằng:

> Với một tập hợp tĩnh S gồm n khóa, luôn có thể xây dựng một bảng băm có thời gian truy cập chính xác $O(1)$ trong trường hợp xấu nhất, và chỉ sử dụng bộ nhớ $O(n)$.

Họ đưa ra ý tưởng **băm hai cấp (two-level hashing)** như sau:

- **Cấp 1:** Sử dụng một hàm băm ngẫu nhiên từ một họ hàm băm phổ quát (universal hash family) để chia các phần tử vào n bucket.
- **Cấp 2:** Với mỗi bucket chứa n_i phần tử, xây dựng một bảng băm con có kích thước n_i^2, sử dụng một hàm băm riêng để đảm bảo không có xung đột trong bucket đó.

Phân tích xác suất trong công trình này cho thấy:

- Xác suất xây dựng bảng băm thành công (không có xung đột ở cấp 2) là cao.
- Tổng bộ nhớ sử dụng cho các bảng cấp 2 là:

$$\sum_i n_i^2 = O(n)$$

Đến hiện tại, **Perfect Hashing** được ứng dụng rộng rãi trong:

- Cơ sở dữ liệu và bảng ký hiệu (symbol table) trong trình biên dịch.
- Hệ thống mạng (router) với yêu cầu tra cứu địa chỉ cực nhanh.
- Phân tích mã ngược, tối ưu máy học, và nén dữ liệu.
- Các hệ thống chỉ đọc (read-only), nơi cấu trúc dữ liệu không thay đổi sau khởi tạo.

3. Giới hạn báo cáo

Bài báo cáo có kiến thức giới hạn trong kỹ thuật **Perfect Hashing** sử dụng *Universal Hashing* dựa theo nội dung của tài liệu `Universal & Perfect Hashing` [3] do thầy Lê Nhựt Nam đưa ra. Lí thuyết được sử dụng từ quá trình học tập bộ môn **Cấu trúc dữ liệu và giải thuật** và tham khảo thêm các nguồn như từ sách `Introduction to Algorithms 3rd Edition 2009` [1], trang Geeks for Geeks, cùng một số nơi khác.
Nội dung của dữ liệu đầu vào giới hạn trong kiểu dữ liệu tĩnh với tập khóa S được biết trước và không thay đổi sau khi xây dựng hàm băm.
Độ dài của chuỗi khóa được giới hạn trong 100 kí tự đầu, các chuỗi khóa lớn hơn có bị bỏ qua.

4. Cấu trúc báo cáo

Báo cáo gồm các phần chính: (i) giới thiệu tổng quan đồ án, (ii) tổng hợp kiến thức nền tảng, (iii) Chi tiết thuật toán áp dụng, (iv) cài đặt và đo kết quả thực nghiệm, và (v) kết luận cùng hướng phát triển cho đồ án.

5. Bảng phân công

MSSV	**Họ và tên**	**Nội dung thực hiện**
24120059	Trần Kim Hữu	Tìm hiểu lý thuyết Perfect Hasing, viết, chỉnh sửa thuật toán và cài đặt chương trình.
24120041	Phạm Võ Đức	Viết báo cáo, tìm hiểu lý thuyết Hashing, Hash Table và tổng hợp tài liệu.
24120006	Đào Thanh Phong	Thực thi chương trình, tạo và xử lý dữ liệu đầu vào, đo thực nghiệm và tìm hiểu lý thuyết Universal Hashing.
24120069	Trần Hoài Bảo Khang	Tìm hiểu lý thuyết độ phức tạp thuật toán, đánh giá thuật toán, thành quả và kết luận tổng quát đồ án.

Bảng I.1: Bảng phân công công việc trong nhóm

CHƯƠNG II

MỘT SỐ HIỂU BIẾT NỀN TẢNG

1. Hashing

Hashing (băm) là một kỹ thuật ánh xạ một khóa (key) từ tập dữ liệu đầu vào sang một chỉ số trong bảng (mảng) để lưu trữ hoặc tra cứu nhanh chóng. Mục tiêu chính của hashing là thực hiện các thao tác như tra cứu, chèn hoặc xoá phần tử trong thời gian trung bình $O(1)$, tức là không phụ thuộc vào kích thước của dữ liệu.

1.1. Nguyên lý cơ bản

Ý tưởng của hashing là sử dụng một **hàm băm** $h(x)$ để ánh xạ một khóa x (thường là chuỗi, số nguyên, v.v.) đến một vị trí trong bảng băm có kích thước m:

$$\text{index} = h(x) \mod m$$

Với mỗi phần tử x, hàm băm sẽ sinh ra một chỉ số trong khoảng $[0, m-1]$ và phần tử đó sẽ được lưu ở vị trí tương ứng trong bảng.

1.2. Các thành phần của Hashing

Hashing gồm bốn thành phần cơ bản:

- **Hash Table**: Cấu trúc dữ liệu dùng để lưu trữ các phần tử đã băm.
- **Hash Function**: Hàm dùng để ánh xạ khóa đầu vào thành chỉ số trong bảng.
- **Collision**: Tình huống xảy ra khi hai khóa khác nhau có cùng giá trị băm.
- **Collision Resolution Techniques**: Các kỹ thuật xử lý xung đột nhằm đảm bảo tính toàn vẹn và hiệu suất.

1.3. Hash Table

Hash Table là một cấu trúc dữ liệu dạng mảng được dùng để lưu trữ các phần tử đã được ánh xạ (băm) từ khóa đầu vào. Kích thước bảng thường được chọn là một số nguyên tố hoặc lũy thừa của 2 nhằm phân phối chỉ số đều hơn, hạn chế xung đột.

Mỗi ô trong bảng có thể chứa:

- Một giá trị duy nhất (nếu không có xung đột)
- Một danh sách liên kết các phần tử (chaining)
- Một con trỏ hoặc cặp key–value

Bảng băm hỗ trợ các thao tác cơ bản với thời gian trung bình $O(1)$:

- Insert (Thêm phần tử)
- Search (Tìm kiếm phần tử)
- Delete (Xoá phần tử)

1.4. Hash Function

Hash Function (hàm băm) là một công thức hoặc thuật toán nhận đầu vào là một khóa (key) và trả về một chỉ số trong bảng băm. Một hàm băm hiệu quả cần đảm bảo:

- Phân phối đều các khóa vào các chỉ số trong bảng
- Tính toán nhanh và đơn giản
- Tránh sinh ra cùng giá trị băm cho các khóa khác nhau (giảm xung đột)

Ví dụ: $h(x) = x \bmod m$.

Một số hàm băm phổ biến:

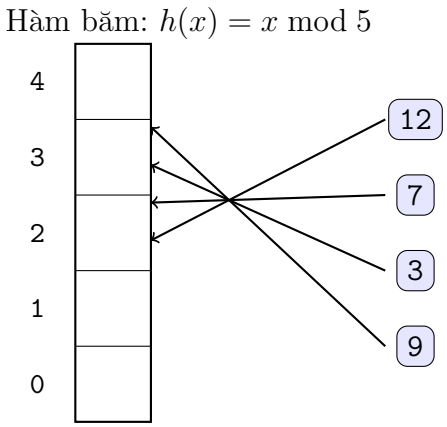
- **Hàm băm chuỗi:** Dựa trên công thức:

$$h(x) = (a_0 + a_1 \cdot p + a_2 \cdot p^2 + ... + a_n \cdot p^n) \mod m$$

trong đó a_i là mã ASCII của ký tự thứ i trong chuỗi, p là số nguyên lớn.

- **Hàm băm hỗn hợp:** Kết hợp các phép toán như XOR, shift, nhân để tăng tính ngẫu nhiên và giảm xung đột.

1.5. Sơ đồ minh họa hashing cơ bản



Hình II.1: Minh họa hàm băm và xung đột khi $h(x) = x \bmod 5$

1.6. Vấn đề xung đột (Collision)

Một trong những thách thức lớn của hashing là **xung đột** (collision): khi hai khóa khác nhau $x \neq y$ lại có cùng giá trị băm, tức là $h(x) = h(y)$. Khi đó, cả hai khóa sẽ được ánh xạ đến cùng một vị trí trong bảng, dẫn đến việc phải xử lý đặc biệt để tránh mất dữ liệu.

Trong ví dụ ở Hình II.1, ta thấy cả `12` và `7` đều được ánh xạ đến ô chỉ số `2`, gây ra xung đột.

1.7. Các phương pháp xử lý xung đột phổ biến

Có nhiều cách để xử lý xung đột khi sử dụng hashing, bao gồm:

- **Chaining (Băm liên kết)**: Mỗi ô trong bảng chứa một danh sách liên kết để lưu nhiều phần tử. Khi có xung đột, phần tử mới được thêm vào danh sách tại vị trí đó.
- **Linear Probing**: Khi xảy ra xung đột, thuật toán kiểm tra các ô tiếp theo trong bảng (tăng tuần tự) cho đến khi tìm được ô trống.
- **Quadratic Probing**: Cũng kiểm tra các ô kế tiếp, nhưng khoảng cách tăng theo bình phương: i^2 thay vì i.

- **Double Hashing**: Dùng một hàm băm thứ hai để xác định bước nhảy tiếp theo, tránh được cụm (clustering).

Các kỹ thuật này giúp giảm thiểu tác động của xung đột, nhưng không đảm bảo truy xuất luôn luôn là $O(1)$ nếu số lượng phần tử tăng quá lớn so với kích thước bảng.

1.8. Vai trò của Hashing trong thực tế

Hashing được ứng dụng rộng rãi trong các lĩnh vực:

- **Cấu trúc dữ liệu:** Hash Table, Hash Map, Hash Set.
- **Trình biên dịch:** lưu trữ và tra cứu bảng ký hiệu (symbol table).
- **Cơ sở dữ liệu:** tổ chức chỉ mục, tra cứu khoá chính.
- **Bảo mật:** mã hóa mật khẩu, chữ ký số (hash function không đảo ngược).
- **Thuật toán:** phân nhóm, kiểm tra trùng lặp, tìm kiếm chuỗi.

1.9. Giới thiệu Perfect Hashing

Trong một số tình huống đặc biệt, khi tập khóa là **tĩnh** (không thay đổi trong quá trình sử dụng), ta có thể xây dựng một bảng băm mà **không xảy ra xung đột nào**. Kỹ thuật này được gọi là **Perfect Hashing** (băm hoàn hảo), cho phép truy xuất trong thời gian $O(1)$ cả trong trường hợp xấu nhất.

2. Perfect Hashing

Perfect Hashing (băm hoàn hảo) là một hàm băm cho ra độ phức tạp khi thực hiện tìm kiếm là $O(1)$ ngay cả khi trong trường hợp xấu nhất khi tập khóa là "tĩnh": mỗi khóa được lưu trong bảng, tập khóa không bao giờ thay đổi. Một số ứng dụng của Perfect Hashing: từ điển từ khóa trong trình biên dịch, tra cứu dữ liệu khi tập khóa không thay đổi, ...
Để tạo ra **Perfect Hashing**, chúng ta sử dụng băm hai cấp, với kĩ thuật Universal Hashing ở mỗi cấp độ. Trong đồ án này, chúng ta thiết kế **Perfect Hashing** theo phương pháp sau:

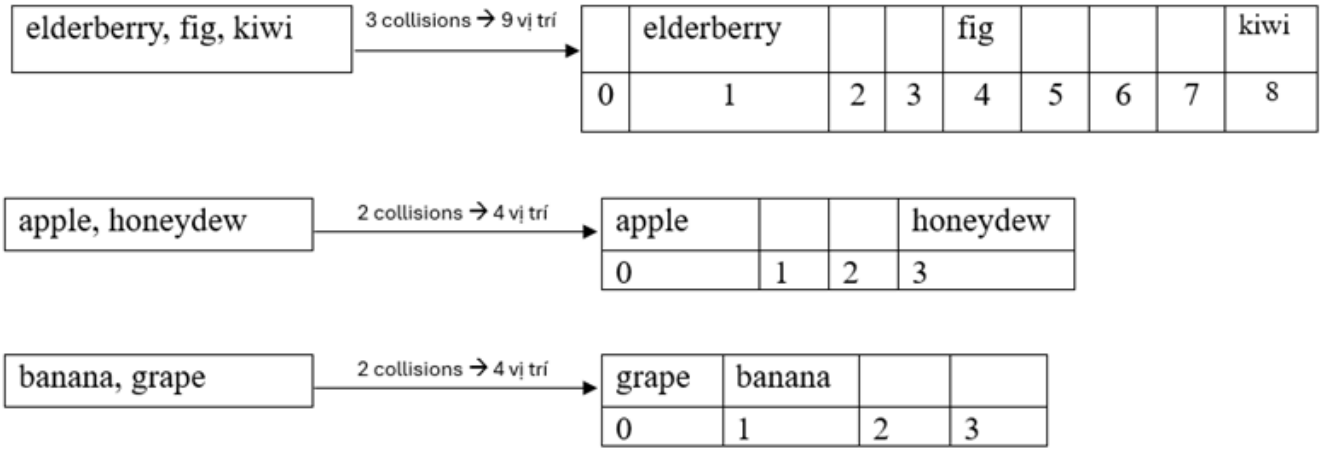
Cấp độ một, chúng ta băm n keys vào m vị trí (m là số nguyên tố nhỏ nhất lớn hơn n) bằng kĩ thuật Universal Hashing.

Cấp độ hai, chúng ta có k collisions ở mỗi vị trí index i từ bảng băm của cấp độ một. Băm k keys xảy ra collisions tại index i đó vào k^2 vị trí sao cho không xảy ra bất kì collision nào ở cấp độ này bằng cách lựa chọn hàm băm thích hợp từ Universal Hashing.

Ví dụ: Tập khóa = {apple, banana, cherry, date, elderberry, fig, grape, honeydew, jackfruit, kiwi}, $n = 10 \rightarrow$ bảng băm kích cỡ $m = 11$.

index	**keys**
0	cherry
1	date
2	
3	elderberry, fig, kiwi
4	
5	
6	
7	apple, honeydew
8	
9	banana, grape
10	jackfruit

Hình II.2: Băm cấp 1



Hình II.3: Băm cấp 2

index	0	1	2	3	4	5	6	7	8
0	cherry								
1	date								
2									
3		elderberry			fig				kiwi
4									
5									
6									
7	apple			honeydew					
8									
9	grape	banana							
10	jackfruit								

Hình II.4: Bảng băm thu được

3. Universal Hashing

3.1. Giới thiệu

Universal Hashing (Băm phổ quát) là một phương pháp chọn ngẫu nhiên hàm băm từ một họ hàm băm sao cho xác suất xảy ra xung đột giữa hai khóa bất kỳ là nhỏ. Mục tiêu là giảm xung đột trong quá trình băm, đặc biệt trong tình huống xấu nhất, bằng cách tận dụng xác suất thay vì cố định một hàm băm cụ thể.

Một họ hàm băm H được gọi là **universal** nếu với mọi hai khóa phân biệt $x \neq y$, xác suất để chúng bị ánh xạ vào cùng một vị trí là nhỏ:

$$\Pr[h(x) = h(y)] \leq \frac{1}{m}$$

Trong đó:

- m: kích thước bảng băm
- h: hàm được chọn ngẫu nhiên từ họ H

3.2. Vai trò trong Perfect Hashing

Trong **Perfect Hashing**, đặc biệt là phương pháp FKS (Fredman, Komlós, Szemerédi), Universal Hashing đóng vai trò quan trọng ở cả hai cấp băm:

- **Cấp 1**: Dùng một hàm băm phổ quát để phân bố n khóa vào n ô của bảng chính, sao cho số lượng khóa trong mỗi ô là ngẫu nhiên và có phân phối tốt.
- **Cấp 2**: Với mỗi ô có nhiều hơn một khóa, một bảng phụ được tạo ra, dùng một hàm băm riêng cũng từ một họ hàm băm phổ quát để đảm bảo không có xung đột trong bảng phụ.

Tính chất ngẫu nhiên của Universal Hashing giúp đảm bảo:

- Trung bình tổng không gian cho các bảng phụ là $O(n)$
- Xác suất cần xây dựng lại bảng là rất thấp (dưới 1/2)
- Kỳ vọng chỉ cần khoảng 1–2 lần thử là sẽ tìm được hàm băm phù hợp

3.3. Độ phức tạp thời gian trong xây dựng bảng băm (Build time)

Cấu trúc hai cấp (Two-level Hashing)

Perfect Hashing sử dụng hai cấp hàm băm:

- **Cấp 1 (Level 1)**: Dùng một hàm băm phổ quát để phân bố n khóa vào $m = n$ ô.
- **Cấp 2 (Level 2)**: Với mỗi ô i có n_i khóa, ta xây dựng bảng phụ kích thước n_i^2 sao cho không có xung đột trong ô đó.

Thời gian trung bình (Expected time)

- Cấp 1:
  - Mỗi phép băm là $O(1)$, tổng cộng $O(n)$ thời gian để băm tất cả khóa
- Cấp 2:
  - Với mỗi ô có n_i khóa, xây dựng bảng phụ mất $O(n_i^2)$ thời gian (tìm hàm băm con không có xung đột)
  - Tổng thời gian là $O\left(\sum n_i^2\right)$

Dùng Universal Hashing:

$$\mathbb{E}\left[\sum n_i^2\right] \leq 2n \Rightarrow \text{Do đó thời gian trung bình để xây hàm băm cấp 2 là } O(n)$$

Trường hợp xấu nhất

Nếu một ô chứa gần hết các khóa, ví dụ $n_i \approx n$, thì bảng phụ phải có kích thước n^2 và thời gian xây dựng là $O(n^2)$.
Tuy nhiên, xác suất xảy ra điều này là rất nhỏ, nhờ tính ngẫu nhiên của Universal Hashing. Dựa vào bất đẳng thức Markov, ta có:

$$\Pr[X \geq a] \leq \frac{\mathbb{E}[X]}{a}$$

Vì vậy, khả năng phải thử lại để chọn một hàm băm khác là thấp, và kỳ vọng số lần thử nhỏ hơn 2.

Kết luận:

Mức độ	**Thời gian xây dựng bảng băm**
Trung bình (Expected)	$O(n)$
Trường hợp xấu nhất	$O(n^2)$ (xác suất thấp)

3.4. Ưu điểm khi dùng Universal Hashing

- **Truy xuất luôn** $O(1)$: Với cấu trúc hai cấp, việc chọn hàm băm từ họ phổ quát đảm bảo thời gian hằng định.
- **Thời gian và không gian trung bình tốt**: Trung bình là $O(n)$ cả về thời gian và không gian.

- **Tránh trường hợp tệ nhất thường xuyên**: Xác suất xảy ra trường hợp xấu là rất thấp.

3.5. Tổng kết

Thành phần	**Vai trò của Universal Hashing**
Cấp 1	Phân bố khóa đều, giảm số lượng xung đột
Cấp 2	Tìm hàm băm không xung đột cho mỗi bảng phụ
Xác suất thất bại	1/2, nhờ bất đẳng thức Markov
Số lần thử trung bình	≤ 2 để chọn được hàm phù hợp
Không gian kỳ vọng	$O(n)$, nhờ $\mathbb{E}[\sum n_i^2] \leq 2n$

4. Độ phức tạp thuật toán

4.1. Giới thiệu

Perfect Hashing (băm hoàn hảo) là một hàm băm có độ phức tạp tìm kiếm là $O(1)$ ngay cả trong trường hợp xấu nhất, khi tập khóa là "tĩnh" (tức là không thay đổi sau khi xây dựng). Một số ứng dụng: từ điển từ khóa trong trình biên dịch, tra cứu dữ liệu không thay đổi,...

Báo cáo này đánh giá độ phức tạp thời gian và không gian dựa trên phương pháp FKS (Fredman, Komlós, Szemerédi) và lý thuyết Universal Hashing.

4.2. Độ phức tạp thời gian truy xuất (Lookup time)

Vì sử dụng cấu trúc hàm băm hai cấp nên thời gian truy xuất dữ liệu luôn luôn là O(1), kể cả trong trường hợp xấu nhất. Kết quả này có được là do khi sử dụng hàm băm hai cấp, mỗi cấp đều đảm bảo không xuất hiện xung đột ở cấp 2.

Chứng minh:

Để tra cứu một khóa x:

- Tính $i = h(x)$ với h là hàm băm cấp 1 $\Rightarrow O(1)$.
- Tính $h_i(x)$ để tìm phần tử $A_i[h_i(x)]$ trong bảng phụ. Vì bảng phụ được xây dựng không có xung đột $\Rightarrow O(1)$.

$\Rightarrow$ Thời gian truy xuất dữ liệu luôn là O(1).

4.3. Độ phức tạp thời gian xây dựng bảng băm (Build time)

Thời gian trung bình để xây dựng bảng băm là O(n), nhưng trong trường hợp xấu nhất có thể đạt $O(n^2)$.

Chứng minh:

Xây dựng hàm băm cấp 1:

- Dùng hàm băm phổ quát $h \in H$ để ánh xạ n khóa vào bảng $m = n$: $O(n)$ (mỗi phép băm là $O(1)$).
- Mỗi ô i có n_i khóa, với $\sum n_i = n$.

Xây dựng hàm băm cấp 2:

Với ô i có n_i khóa:

- Tạo bảng phụ A_i có kích thước $O(n_i^2)$.
- Thời gian xây dựng mỗi A_i là $O(n_i^2)$.

Tổng thời gian xây dựng hàm băm cấp 2:

$$O\left(\sum n_i^2\right)$$

Kỳ vọng:

- $\mathbb{E}[\sum_i n_i^2] = \mathbb{E}[\sum_x \sum_y C_{xy}]$ ($C_{xy} = 1$ nếu x và y va chạm, ngược lại thì $C_{xy} = 0$)

  $= N + \sum_x \sum_{y \neq x} \mathbb{E}[C_{xy}]$

  $\leq N + N(N-1)/M$ (với $\frac{1}{M}$ nằm trong định nghĩa của Universal Hashing)

  $< 2N$ (vì $M = N$)

- Dựa theo bất đẳng thức Markov, ta có xác suất thất bại:

$$\Pr\left[\sum_i n_i^2 \geq 4N\right] \leq \frac{\mathbb{E}[\sum_i n_i^2]}{4N} \leq \frac{2N}{4N} = \frac{1}{2}$$

Do đó, xác suất thành công $> \frac{1}{2}$

- Khi thất bại, số lần thử cần để tìm được hàm h phù hợp là $\leq \frac{1}{p} = \frac{1}{1/2} = 2$.

  $\Rightarrow$ Vậy chỉ cần khoảng 1 hoặc 2 lần thử là tìm được hàm h phù hợp làm cho

$$\sum_i n_i^2 \leq 4N$$

- **Khi đó, thời gian xây dựng bảng băm cấp 2 là** $O(n)$.

Trường hợp xấu nhất: Nếu $n_i \approx n \Rightarrow O(n^2)$, nhưng xác suất rất thấp ($\leq \frac{1}{2}$).

Kết luận: Thời gian xây dựng trung bình là $O(n)$, còn trong trường hợp xấu nhất là $O(n^2)$ (xác suất rất thấp).

4.4. Độ phức tạp không gian

Cấp 1: Kích thước bảng băm $m = O(n)$

Cấp 2: Tương tự cách chứng minh ở **4.3**. Với tổng không gian là $O(\sum n_i^2)$

Theo Universal Hashing:

$$\mathbb{E}\left[\sum n_i^2\right] \leq 2n$$

Dựa theo bất đẳng thức Markov:

$$\Pr\left[\sum n_i^2 \geq 4n\right] \leq \frac{1}{2}$$

$\Rightarrow$ Không gian trung bình O(n) khi $\Pr\left[\sum n_i^2 > 4n\right] \leq \frac{1}{2}$.

Tổng: $O(n) + O(n) + O(n) = O(n)$.

Trường hợp xấu nhất: $O(n^2)$ nếu $n_i \approx n$, nhưng xác suất rất thấp ($\leq \frac{1}{2}$).

Kết luận: Không gian xây dựng trung bình là $O(n)$, còn trong trường hợp xấu nhất là $O(n^2)$ với xác suất ($\leq \frac{1}{2}$).

4.5. Tóm tắt

Độ phức tạp	**Trung bình**	**Trường hợp xấu nhất**
Tìm kiếm	$O(1)$	$O(1)$
Xây dựng	$O(n)$ (kỳ vọng)	$O(n^2)$ (hiếm gặp)
Không gian	$O(n)$	$O(n^2)$ (hiếm gặp)

CHƯƠNG III

CHI TIẾT THUẬT TOÁN

1. Dữ liệu đầu vào và dữ liệu đầu ra

1.1. Input

Các tập dữ liệu đầu vào là các tệp văn bản (Original_1.txt, Original_2.txt, Original_3.txt), mỗi dòng chứa một mục văn bản. Tập khóa được trích xuất từ đầu dòng, chỉ giữ lại tối đa 100 ký tự đầu tiên và lọc bỏ ký tự ngoài mã ASCII chuẩn (0–127).
Độ lớn tập dữ liệu được thay đổi từ 2.250 đến 370.000 dòng để đo hiệu năng thuật toán theo từng quy mô. Mỗi lần thử nghiệm chọn 5 từ khóa đại diện để tra cứu và đo thời gian truy xuất trung bình.

1.2. Output

Sau khi xây dựng bảng băm, chương trình thực hiện tra cứu cho mỗi từ khóa đầu vào được nhập từ bàn phím. Mỗi từ được tra cứu lặp lại 10^6 lần nhằm tính thời gian trung bình một lần truy xuất, kết quả gồm:

- Dòng dữ liệu (val) tương ứng nếu tìm thấy khóa.
- "Not found" nếu khóa không tồn tại trong tập dữ liệu.
- Thời gian trung bình tra cứu một từ (tính bằng đơn vị picosecond).

Thời gian xây dựng bảng băm (gồm cấp 1 và cấp 2) được đo bằng đơn vị milliseconds. Kết quả được tổng hợp trong bảng đo thực nghiệm để đánh giá hiệu năng hệ thống.

2. Mã giả (Pseudo code)

2.1. Mã giả của hàm băm trong tập Universal Hashing

```
Function UNIVERSAL-HASH(key: string, setA: array of integers, mod: integer) → integer
```

```
    sum ← 0
    len ← MIN(length of key, length of setA)
    for i ← 0 to len - 1 do
        val ← ASCII code of key[i]
        if val < 0 or val > 127 then
            return -1
        end if
        sum ← sum + (val × setA[i])
    end for
    return sum mod mod
End Function
```

2.2. Mã giả hàm xây dựng bảng băm hoàn hảo

```
Function BUILD(path: string)
    data ← empty list of Value
    READ-FILE(path, data)
    REMOVE-DUPLICATE-KEYS(data)
    n ← size of data
    m ← FIND-SMALLEST-PRIME-GREATER-THAN(n)
    table ← array of m empty lists
    // Generate random setAHash1 of length 100
    setAHash1 ← array of 100 random integers in [0, m - 1]
    // First-level hashing
    for each element cur in data do
        index ← UNIVERSAL-HASH(cur.key, setAHash1, m)
        if index < 0 then
            continue
        end if
        PUSH_BACK cur to table[index]
    end for
    // Prepare for second-level hashing
```

```
hasLevel2Hash ← array of m false values
setAHash2 ← array of m arrays
for i ← 0 to m - 1 do
    sizeCollision ← size of table[i]
    if sizeCollision < 2 then
        continue
    end if
    hasLevel2Hash[i] ← true
    sizeTable2 ← sizeCollision × sizeCollision
    res ← array of sizeTable2 pairs (Value, false)
    // Repeat until no collision in second-level hashing
    repeat
        collision ← false
        RESET res to all (empty Value, false)
        // Generate random setAHash2[i] of length 100
        setAHash2[i] ← array of 100 random integers in [0, sizeTable2 - 1]
        for each element s in table[i] do
            index2 ← UNIVERSAL-HASH(s.key, setAHash2[i], sizeTable2)
            if res[index2].second = true then
                collision ← true
                break
            end if
            res[index2] ← (s, true)
        end for
    until collision = false
    // Save second-level hash table
    table[i] ← array of sizeTable2
    for j ← 0 to sizeTable2 - 1 do
        table[i][j] ← res[j].first
    end for
end for
```

```
End Function
```

2.3. Mã giả hàm tìm kiếm trong Perfect Hashing

```
Function SEARCH(key: string) → string
    index1 ← UNIVERSAL-HASH(key, setAHash1, size of table)
    if index1 is out of bounds OR table[index1] is empty then
        return "Not found"
    end if
    if hasLevel2Hash[index1] = false then
        // No second-level hashing, check directly
        if first element in table[index1] has key equal to key then
            return its value
        else
            return "Not found"
        end if
    end if
    // Use second-level hashing
    index2 ← UNIVERSAL-HASH(key, setAHash2[index1], size of table[index1])
    if index2 is out of bounds OR table[index1][index2].key != key then
        return "Not found"
    end if
    return table[index1][index2].val
End Function
```

3. Phân tích độ phức tạp

3.1. Hàm UNIVERSAL-HASH

Hàm này thực hiện một phép cộng có trọng số trên các ký tự của khóa.

- Đầu vào: chuỗi khóa có độ dài k.
- Độ dài của mảng hệ số A cố định là $c = 100$.

Độ phức tạp thời gian:

$$T_{\text{hash}}(k) = \mathcal{O}(\min(k, c)) = \mathcal{O}(1) \quad (\text{vì } c \text{ là hằng số})$$

3.2. Hàm `BUILD` – Xây dựng bảng băm hoàn hảo

Băm cấp 1:

- Có n phần tử.
- Mỗi phần tử được băm bằng `UNIVERSAL-HASH` $\rightarrow \mathcal{O}(n)$.

Băm cấp 2:

- Với mỗi bucket i chứa n_i phần tử, tạo bảng phụ kích thước n_i^2.
- Lặp lại chọn hàm băm cấp 2 cho đến khi không có xung đột.

Độ phức tạp trung bình:

$$T_{\text{build}} = \mathcal{O}(n) + \mathcal{O}\left(\sum_{i=1}^{m} n_i^2\right) = \mathcal{O}(n)$$

3.3. Hàm `SEARCH`

- Tính chỉ số cấp 1 bằng `UNIVERSAL-HASH`: $\mathcal{O}(1)$.
- Nếu không có băm cấp 2: so sánh trực tiếp một phần tử $\rightarrow \mathcal{O}(1)$.
- Nếu có băm cấp 2: tính chỉ số cấp 2 và truy cập phần tử trong bảng phụ $\rightarrow \mathcal{O}(1)$.

Độ phức tạp thời gian tìm kiếm:

$$T_{\text{search}} = \mathcal{O}(1) \quad (\text{trong mọi trường hợp, sau khi khởi tạo})$$

3.4. Tổng kết độ phức tạp

Hàm	**Độ phức tạp thời gian**
`UNIVERSAL-HASH`	$\mathcal{O}(1)$
`BUILD`	$\mathcal{O}(n)$
`SEARCH`	$\mathcal{O}(1)$

Bảng III.1: Tổng kết độ phức tạp thời gian của các thành phần chính

CHƯƠNG IV

CÀI ĐẶT THUẬT TOÁN

1. Cấu hình thực nghiệm

- CPU: Intel Core i7-1065G7
- RAM: 20 GB
- Clock Speed: 1.3-3.9 GHz

2. Cách cài

2.1. Ngôn ngữ và công cụ sử dụng

Chương trình được viết bằng ngôn ngữ **C++17**, biên dịch và chạy trên môi trường `g++`. Các thư viện chuẩn được sử dụng bao gồm:

- `<vector>` để lưu bảng băm cấp 1 và cấp 2.
- `<unordered_set>` để lọc các khóa trùng lặp.
- `<random>` để sinh hệ số băm ngẫu nhiên.
- `<chrono>` để đo thời gian thực thi.
- `<fstream>` và `<sstream>` để xử lý tập tin.

2.2. Cấu trúc lớp

Chương trình cài đặt một lớp chính tên là `PerfectHashTable`, bao gồm các thành phần sau:

- `vector<vector<Value» table;` — bảng băm hai cấp.
- `vector<int> setAHash1;` — hệ số của hàm băm cấp 1.

- `vector<vector<int» setAHash2;` — hệ số của các hàm băm cấp 2.
- `vector<bool> hasLevel2Hash;` — cờ đánh dấu bucket dùng cấp 2.
- `int m;` — số lượng bucket, là số nguyên tố gần nhất $\geq n$.

2.3. Khởi tạo value

```
struct Value {
    string key;
    string val;
};
```

2.4. Các bước cài đặt chính

Đọc dữ liệu và loại bỏ khóa trùng

Dữ liệu được đọc từ file văn bản (mỗi dòng là một mục), sau đó lọc trùng bằng `unordered_set`:

```
void readFile(const string& path, vector<Value>& vec) {
    ifstream ifs(path);
    string temp;
    Value cur;
    while (getline(ifs, temp)) {
        if (temp.empty() || temp.size() < 3) continue;
        stringstream ss(temp);
        ss >> cur.key;
        cur.val = temp;
        vec.push_back(cur);
    }
}

void removeDuplicateKeys(vector<Value>& data) {
    unordered_set<string> seen;
    vector<Value> filtered;
    for (const auto& val : data) {
```

```
        if (seen.count(val.key)) continue;
        seen.insert(val.key);
        filtered.push_back(val);
    }
    data = move(filtered);
}
```

Hàm băm ***universalHash***

Hàm `universalHash()` là thành phần trung tâm của cả hai cấp băm. Hàm này nhận vào một chuỗi khóa, một mảng hệ số ngẫu nhiên và số lượng phần tử trong bảng, sau đó tính tổng có trọng số và lấy modulo:

- Nếu khóa chứa ký tự ngoài bảng mã ASCII $0-127$, hàm trả về -1.
- Nếu độ dài khóa lớn hơn độ dài mảng hệ số (mặc định là 100), chỉ xét 100 ký tự đầu.

```
int universalHash(const string& key, const vector<int>& setA, int mod) {
    int sum = 0;
    int len = min((int)key.size(), (int)setA.size());
    for (int i = 0; i < len; ++i) {
        int val = (int)key[i];
        if (val < 0 || val > 127) return -1;
        sum += val * setA[i];
    }
    return sum % mod;
}
```

Tạo bảng băm cấp 1

Chọn số nguyên tố gần nhất $m \geq n$ làm kích thước bảng, sau đó phân phối các phần tử vào bảng theo hàm băm:

```
m = findSmallestPrimeGreaterThan(n);
table.assign(m, vector<Value>());
```

```
3
4  setAHash1.resize(100);
5  uniform_int_distribution<> dis(0, m - 1);
6  for (int& num : setAHash1) {
7      num = dis(gen);
8  }
9
10 for (const auto& cur : data) {
11     int index = universalHash(cur.key, setAHash1, m);
12     if (index < 0) continue;
13     table[index].push_back(cur);
14 }
```

Tạo bảng băm cấp 2

Với các bucket có nhiều hơn 1 phần tử, tạo bảng băm phụ kích thước $n_i^2 \times k$ với các các bậc k từ mức $1 \rightarrow 6$ và là lặp đến khi không còn xung đột, , ở đây lựa bậc 4 để tối ưu, nguyên nhân là trong thực nghiệm vấn đề dữ liệu rất lớn, bậc 1 vẫn còn gây lỗi nên ta phải phát sinh thêm:

```
1  for (int i = 0; i < m; ++i) {
2      int sizeCollision = table[i].size();
3      if (sizeCollision < 2) continue;
4      hasLevel2Hash[i] = true;
5      int sizeTable2 = sizeCollision * sizeCollision * 4;
6      vector<pair<Value, bool>> res(sizeTable2);
7      uniform_int_distribution<> dis2(0, sizeTable2 - 1);
8      bool collision;
9      do {
10         collision = false;
11         fill(res.begin(), res.end(), make_pair(Value{ "", "" }, false));
12         setAHash2[i].resize(100);
13         for (int& num : setAHash2[i]) {
14             num = dis2(gen);
15         }
```

```
        for (const auto& s : table[i]) {
            int index2 = universalHash(s.key, setAHash2[i], sizeTable2);
            if (res[index2].second) {
                collision = true;
                break;
            }
            res[index2] = { s, true };
        }
    } while (collision);

    table[i].clear();
    table[i].resize(sizeTable2);
    for (int j = 0; j < sizeTable2; j++) {
        table[i][j] = res[j].first;
    }
}
```

Tìm kiếm

Hàm `search()` kiểm tra băm cấp 1, nếu cần thì tiếp tục băm cấp 2 để tìm đúng chỉ số:

```
string search(const string& key) {
    int index1 = universalHash(key, setAHash1, table.size());
    if (index1 >= table.size() || table[index1].empty()) return "Not
       found";
    if (!hasLevel2Hash[index1]) {
        if (table[index1].begin()->key == key) return table[index1].
           begin()->val;
        return "Not found";
    }
    int index2 = universalHash(key, setAHash2[index1], table[index1].
       size());
    if (index2 >= table[index1].size() || table[index1][index2].key !=
       key) return "Not found";
```

```
    return table[index1][index2].val;
}
```

2.5. Đo thời gian tìm kiếm

Trong hàm `main()`, mỗi từ khóa được tra cứu 10^6 lần và tính thời gian trung bình:

```
const int repeat = 1000000;
for (const string& s : vec) {
    start = chrono::high_resolution_clock::now();
    string temp;
    for (int i = 0; i < repeat; ++i) {
        temp = pht.search(s);
    }
    end = chrono::high_resolution_clock::now();
    auto duration_ns = chrono::duration_cast<chrono::nanoseconds>(end -
       start);
    long long duration_ps = duration_ns.count() * 1000;
    cout << "Avg time: " << duration_ps / repeat << " ps" << endl;
}
```

3. Kết quả đo đạt và đánh giá

Thông qua thực thi các tài liệu đầu vào ta có bảng thời gian hashing và tìm kiếm dữ liệu của *Perfect Hashing* với các bậc k của bảng băm cấp 2 ($n^2 \times k$) như sau:

Hashing của File: Original_1.txt (Bậc 4)					
Độ lớn file (Số dòng) / **Word**	**3600**	**7200**	**14400**	**36000**	**72000**
Word 1	152	198	127	191	146
Word 2	142	131	159	121	157
Word 3	153	154	156	165	172
Word 4	160	134	163	150	175
Word 5	167	163	159	184	160
Thời gian trung bình	155	156	153	162	162
Thời gian Hashing	10ms	22ms	57ms	116ms	142ms

Bảng IV.1: Kết quả Hashing cho File Original_1.txt với bậc 4

Hashing của File: Original_2.txt (bậc 4)					
Độ lớn file (Số dòng) / **Word**	**18500**	**37000**	**74000**	**185000**	**370000**
Word 1	125	144	160	168	141
Word 2	139	153	173	141	136
Word 3	133	163	100	117	101
Word 4	126	120	117	105	126
Word 5	145	126	137	119	151
Thời gian trung bình	134	141	137	130	131
Thời gian Hashing	66ms	143ms	246ms	663ms	1277ms

Bảng IV.2: Kết quả Hashing cho File Original_2.txt với bậc 4

Thời gian Hashing của File: Original_3.txt (bậc 4)					
Độ lớn file (Số dòng) / Số words	**2250**	**4500**	**9000**	**22500**	**45000**
Word 1	136	130	98	100	110
Word 2	110	103	137	138	116
Word 3	118	117	140	106	141
Word 4	115	130	125	141	97
Word 5	125	112	126	128	146
Thời gian trung bình	121	118	125	123	122
Thời gian Hashing	11ms	15ms	31ms	57ms	125ms

Bảng IV.3: Kết quả Hashing cho File Original_3.txt với bậc 4

So sánh kết quả của file Original_3 ở các bậc khác:

Hashing của File: Original_3.txt (Bậc 1)					
Độ lớn file (Số dòng) / Word	**2250**	**4500**	**9000**	**22500**	**45000**
Word 1	89	128	127	none	none
Word 2	122	130	131	none	none
Word 3	142	98	104	none	none
Word 4	150	115	122	none	none
Word 5	137	134	110	none	none
Thời gian trung bình	128	121	119	none	none
Thời gian Hashing	34ms	43ms	62ms	none	none

Bảng IV.4: Kết quả Hashing cho File Original_3.txt với bậc 1

Hashing của File: Original_3.txt (Bậc 2)					
Độ lớn file (Số dòng) / **Word**	**2250**	**4500**	**9000**	**22500**	**45000**
Word 1	88	130	116	114	118
Word 2	125	137	128	124	127
Word 3	145	171	95	92	145
Word 4	148	100	152	95	103
Word 5	133	102	133	130	134
Thời gian trung bình	128	128	125	111	125
Thời gian Hashing	18ms	31ms	43ms	74ms	168ms

Bảng IV.5: Kết quả Hashing cho File Original_3.txt với bậc 2

Hashing của File: Original_3.txt (Bậc 3)					
Độ lớn file (Số dòng) / **Word**	**2250**	**4500**	**9000**	**22500**	**45000**
Word 1	85	85	128	156	117
Word 2	132	132	107	127	129
Word 3	141	142	107	96	140
Word 4	155	120	177	115	98
Word 5	129	124	157	130	133
Thời gian trung bình	128	121	135	125	123
Thời gian Hashing	19ms	26ms	44ms	72ms	151ms

Bảng IV.6: Kết quả Hashing cho File Original_3.txt với bậc 3

Hashing của File: Original_3.txt (Bậc 5)					
Độ lớn file (Số dòng) / **Word**	**2250**	**4500**	**9000**	**22500**	**45000**
Word 1	85	120	156	124	124
Word 2	121	122	100	139	123
Word 3	139	141	94	95	147
Word 4	145	92	131	100	100
Word 5	127	126	173	125	134
Thời gian trung bình	123	120	131	117	126
Thời gian Hashing	13ms	17ms	32ms	57ms	122ms

Bảng IV.7: Kết quả Hashing cho File Original_3.txt với bậc 5

Hashing của File: Original_3.txt (Bậc 6)					
Độ lớn file (Số dòng) / **Word**	**2250**	**4500**	**9000**	**22500**	**45000**
Word 1	84	86	117	115	123
Word 2	129	131	123	129	137
Word 3	137	142	96	90	140
Word 4	151	107	158	97	99
Word 5	128	136	137	131	122
Thời gian trung bình	126	120	126	112	124
Thời gian Hashing	9ms	20ms	35ms	56ms	120ms

Bảng IV.8: Kết quả Hashing cho File Original_3.txt với bậc 6

4. Nhận xét

4.1. Ưu điểm

- Thời gian tra cứu rất nhanh: có độ phức tạp là $O(1)$ trong mọi trường hợp nhờ cấu trúc băm 2 cấp, không có xung đột ở bảng băm cấp 2. Nhanh hơn so với băm thông thường (open addressing, chaining).
- Tiết kiệm không gian: độ phức tạp không gian mong muốn là $O(n)$, với $E[\Sigma n_i^2] \leq 2n$ (Universal Hashing), $Pr[\Sigma n_i^2 \geq 4n] \leq \frac{1}{2}$ (Markov).
- Xác suất xảy ra va chạm trong bảng băm cấp 2 thấp: xác suất xảy ra ít nhất 1 va chạm là $\leq 1/2$ với số lần thử ≤ 2 để chọn được hàm h không gây va chạm.
- Ứng dụng: là hàm băm lý tưởng để xử lý tập khóa tĩnh, ví dụ như việc tra cứu từ điển.
- Universal Hashing: sử dụng hàm băm phổ quát đảm bảo phân bố đều, giảm xác suất trường hợp xấu so với hàm băm cố định trong bảng băm thông thường.

4.2. Hạn chế

- Yêu cầu tập khóa tĩnh: **Perfect Hashing** chỉ hiệu quả đối với tập khóa cố định, yêu cầu phải tạo lập bảng băm mới khi thay đổi. Trong khi đó băm thông thường và Cuckoo Hashing hỗ trợ thêm/xóa dễ dàng.
- Thời gian xây dựng ban đầu: trung bình tốn thời gian $O(n)$, nhưng trong trường hợp xấu nhất có thể lên đến $O(n^2)$ nếu $n_i \approx n$ (xác suất $\leq \frac{1}{2}$). Trong khi đó Cuckoo Hashing có thời gian chèn trung bình là $O(1)$, còn băm thông thường thì nhanh hơn do không yêu cầu tránh va chạm hoàn toàn.
- Độ phức tạp không gian trong trường hợp xấu: có thể đạt đến $O(n^2)$, trong khi các phương pháp khác như Cuckoo Hashing ($O(n)$ với hai bảng) và băm thông thường ($O(n)$ với chaining) ít gặp trường hợp xấu hơn.
- Phụ thuộc vào việc chọn hàm băm h và h_i: nếu chọn ngẫu nhiên không tốt có thể dẫn đến chi phí cao hơn.

4.3. Thực nghiệm

Thông thường chọn kích thước bảng băm cấp 2 là n^2, tuy nhiên trong thực tế nếu dữ liệu phân phối không đều, kích thước này có thể gây lỗi sinh vô hạn do thiếu vị trí. Do đó chọn kích thước $4n^2$ là hợp lý hơn để tránh lỗi.

- Với bảng phụ cấp 1 (kích thước $\approx n$), hiệu năng giảm mạnh khi quy mô dữ liệu lớn, do xảy ra nhiều xung đột.
- Từ cấp 2 trở lên, thời gian tra cứu cải thiện đáng kể và trở nên ổn định ở mức 120–130 picosecond bất kể kích thước tập dữ liệu.
- Với bảng phụ kích thước bậc 4 ($\approx 4n^2$), hiệu năng đạt mức cao nhất và ổn định nhất. Thời gian xây dựng bảng dao động từ 10ms đến khoảng 140ms, ngay cả khi số lượng dòng lên đến 72.000.

Nhìn chung, cấp độ băm ảnh hưởng trực tiếp đến khả năng loại bỏ xung đột và tối ưu hiệu suất tra cứu. Tuy nhiên, việc chọn bậc quá cao cũng làm tăng chi phí bộ nhớ. Vì vậy, trong thực tiễn, việc lựa chọn bậc phù hợp (thường từ 2 đến 4) sẽ cân bằng tốt giữa hiệu suất và tài nguyên sử dụng.

4.4. So sánh

Tiêu chí	**Perfect Hashing**	**Chaining**	**Open Addressing**	**Cuckoo Hashing**
Thời gian tra cứu	$O(1)$	$O(1)$ trung bình, $O(n)$ xấu nhất	$O(1)$ trung bình, $O(\log n)$ khi đầy	$O(1)$ trung bình
Thời gian xây dựng	$O(n)$ trung bình, $O(n^2)$ xấu nhất	$O(n)$ trung bình	$O(n)$, phụ thuộc độ lấp đầy	$O(n)$, có tái băm
Không gian	Kỳ vọng $O(n)$, xấu nhất $O(n^2)$	$O(n)$ (danh sách liên kết)	$O(n)$ (bảng cố định)	$O(n)$ (hai bảng)

Xác suất va chạm	Xác suất $\leq \frac{1}{2}$ trong bảng băm cấp 2, trung bình ≤ 2 lần thử để tìm	Phụ thuộc vào hàm băm	Cao khi bảng đầy	Thấp, có tái băm
Hỗ trợ dữ liệu động	Không hỗ trợ	Có	Có, nhưng phức tạp khi đầy	Có, dễ thêm/xóa
Ứng dụng	Từ điển, dữ liệu tĩnh	Dữ liệu động	Ứng dụng yêu cầu nhanh	Hệ thống động, tra cứu nhanh
Đặc điểm nổi bật	Không va chạp ở bảng băm cấp 2, hiệu suất ổn định	Đơn giản, linh hoạt	Tiết kiệm không gian	Tra cứu nhanh, hỗ trợ dữ liệu động

CHƯƠNG V

KẾT LUẬN

1. Các kết quả đạt được

Trong quá trình thực hiện đề tài, chúng em đã hoàn thành các mục tiêu đề ra, cụ thể như sau:

- Xây dựng hệ thống băm hoàn hảo hai cấp:
  - Cấp 1: Triển khai hàm băm phổ quát (Universal Hashing) với bảng băm có kích thước $m \times m$ với m là số nguyên tố nhỏ nhất lớn hơn hoặc bằng tổng số phần tử đầu vào.
  - Cấp 2: Phát triển cơ chế tạo bảng băm con tại mỗi vị trí xảy ra va chạm. Kích thước bảng con được thiết lập là s^2 (với s là số lượng phần tử va chạm), đồng thời lựa chọn hàm băm phù hợp để loại bỏ hoàn toàn các va chạm trong bảng con.
- Phát triển các hàm phụ trợ thiết yếu:
  - Triển khai hàm băm phổ quát (Universal Hashing) với khả năng chọn ngẫu nhiên hệ số băm.
  - Xây dựng chức năng kiểm tra số nguyên tố cho việc xác định kích thước bảng băm.
  - Phát triển mô đun xử lý và đọc dữ liệu từ các tập tin đầu vào.
  - Tích hợp công cụ `std::chrono` để đo lường và đánh giá thời gian thực thi của hệ thống.
- Thực hiện thực nghiệm và đo lường hiệu suất:
  - Chạy thử với ba tập dữ liệu khác nhau có kích thước từ 2.000 đến 370.000 dòng.
  - Thực hiện truy vấn lặp lại 10^6 lần cho mỗi từ khóa để đánh giá độ ổn định thời gian truy xuất.
  - Đo và ghi lại thời gian xây dựng bảng và truy xuất tương ứng cho từng tập và cấp băm.
- Phân tích kết quả và đưa ra đánh giá:

  - So sánh hiệu năng của các cấp băm từ bậc 1 đến bậc 6.
  - Đưa ra kết luận về cấp độ tối ưu giữa tốc độ tra cứu và chi phí bộ nhớ (thường là bậc 4).
  - Phát hiện giới hạn khi sử dụng cấp thấp (nhiều xung đột) hoặc cấp cao (tốn bộ nhớ).

2. Những kiến thức thu nhận được

Qua quá trình nghiên cứu và triển khai, chúng em đã tích lũy được những kiến thức chuyên sâu về các khái niệm và kỹ thuật sau:

- Perfect Hashing (Băm hoàn hảo): Nắm vững nguyên lý thiết kế bảng băm không có va chạm, đặc biệt phù hợp cho các tập dữ liệu tĩnh, đảm bảo thời gian truy xuất dữ liệu là O(1).
- Universal Hashing (Băm phổ quát): Hiểu rõ cách lựa chọn ngẫu nhiên một hàm băm từ một họ hàm nhất định để giảm thiểu xác suất xảy ra va chạm, từ đó nâng cao hiệu quả của bảng băm.
- Cấu trúc bảng băm hai cấp: Nắm bắt được kiến trúc và cơ chế hoạt động của bảng băm hai cấp, bao gồm:
  - Bảng chính: Sử dụng hàm băm cấp 1 để phân phối các phần tử.
  - Bảng phụ: Được tạo ra khi có va chạm ở bảng chính, với việc lựa chọn lại hàm băm để giải quyết va chạm hiệu quả.
- Đánh đổi chi phí xây dựng và hiệu suất truy xuất: Nhận thức rõ sự đánh đổi giữa việc chấp nhận chi phí xây dựng bảng băm ban đầu cao hơn để đổi lấy thời gian tra cứu cực nhanh (O(1)) trong các ứng dụng yêu cầu hiệu suất cao.

3. Đánh giá tổng quan

Hệ thống **Perfect Hashing** được triển khai đã thể hiện những ưu điểm và hạn chế nhất định:

- Tính ứng dụng: Kỹ thuật băm hoàn hảo đặc biệt phù hợp cho các ứng dụng đòi hỏi tốc độ tra cứu cao như bảng ký hiệu trong trình biên dịch hoặc các từ điển với tập dữ liệu cố định.
- Hiệu suất tra cứu: Hệ thống hoạt động hiệu quả với các tập dữ liệu tĩnh, đảm bảo thời gian tra cứu trung bình là $O(1)$, đáp ứng yêu cầu về tốc độ.

- Không gian lưu trữ: Không gian bộ nhớ kỳ vọng của hệ thống là $O(n)$. Tuy nhiên, cần lưu ý rằng vẫn có xác suất nhỏ xảy ra trường hợp xấu nhất, dẫn đến không gian bộ nhớ tăng lên $O(n^2)$.
- Thời gian xây dựng: Thời gian xây dựng bảng băm hoàn hảo trung bình là O(n), đủ nhanh và khả thi cho các tập khóa có kích thước vừa phải.

4. Hướng phát triển trong tương lai

Để hoàn thiện và mở rộng khả năng của hệ thống, chúng em mong muốn các hướng phát triển sau:

- Hỗ trợ đa ngôn ngữ và ký tự đặc biệt: Mở rộng khả năng xử lý các bộ ký tự Unicode và các ký tự đặc biệt, thay vì chỉ giới hạn ở ASCII như hiện tại.
- Lưu trữ cấu trúc bảng băm: Phát triển tính năng cho phép lưu trữ cấu trúc bảng băm đã được xây dựng ra tập tin. Điều này sẽ giúp tránh việc phải tái tạo lại bảng băm mỗi khi ứng dụng khởi động, tối ưu hóa thời gian và tài nguyên.
- Nghiên cứu các kỹ thuật băm khác: Tiếp tục nghiên cứu và tìm hiểu sâu hơn về các kỹ thuật băm tiên tiến khác như Cuckoo Hashing, Chaining và Bloom Filter để so sánh, đánh giá và tích hợp nếu phù hợp với các yêu cầu cụ thể.

Bibliography

[1] *Introduction to Algorithms 3rd Edition 2009*. URL: https://drive.google.com/file/d/1kOy_9341U7AvjyUQZOsSuQcVWcgI6k5D. (Date de consultation : 21/5/2025).

[2] *Storing a Sparse Table with O(1) Worst Case Access Time*. URL: https://www.cs.princeton.edu/courses/archive/fall09/cos521/Handouts/storingasparse.pdf. (Date de consultation : 21/5/2025).

[3] *Universal & Perfect Hashing*. URL: https://www.cs.cmu.edu/~avrim/451f11/lectures/lect1004.pdf. (Date de consultation : 21/5/2025).