

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC BÁCH KHOA  
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH



## BÁO CÁO BÀI TẬP LỚN

### LẬP TRÌNH NÂNG CAO

---

Xây dựng Web để tra cứu dữ liệu sao kê của MTTQ VN

---

GVHD: Lê Đình Thuận  
SV thực hiện: Nguyễn Minh Quân – 2212804  
Nguyễn Minh Toàn – 2213533  
Nguyễn Bảo Trâm – 2213572  
Huỳnh Thái Minh Châu – 2210352  
Huỳnh Nga – 2111818  
Lớp: L01 - Nhóm: 498

Tp. Hồ Chí Minh, Tháng 12 / 2024

# Mục lục

<b>1</b>	<b>Tổng quan đề tài</b>	<b>2</b>
1.1	Giới thiệu . . . . .	2
1.2	Phạm vi nghiên cứu . . . . .	2
<b>2</b>	<b>Cấu trúc dữ liệu và giải thuật tìm kiếm</b>	<b>2</b>
2.1	Cấu trúc dữ liệu sử dụng . . . . .	2
2.1.1	Struct Transaction . . . . .	2
2.1.2	Vector (std::vector<Transaction>) . . . . .	2
2.2	Giải thuật tìm kiếm sử dụng . . . . .	3
2.2.1	Tìm kiếm theo số tiền gửi (Hàm creditSearch) . . . . .	3
2.2.2	Tìm kiếm theo nội dung chuyển khoản (Hàm detailSearchParallel) . . . . .	3
<b>3</b>	<b>Công cụ và công nghệ sử dụng</b>	<b>4</b>
3.1	Công nghệ . . . . .	4
3.1.1	Ngôn ngữ C++ . . . . .	4
3.1.2	Thư viện Flask . . . . .	4
3.1.3	Next.js . . . . .	4
3.1.4	Bootstrap . . . . .	4
3.2	Công cụ . . . . .	4
3.2.1	Git . . . . .	4
3.2.2	Flask-REStx . . . . .	5
3.2.3	Swagger UI . . . . .	5
<b>4</b>	<b>Thiết kế giao diện người dùng</b>	<b>5</b>
<b>5</b>	<b>Kết quả</b>	<b>5</b>
5.0.1	Kiểm thử hiệu năng . . . . .	5
5.0.2	Tài nguyên liên quan . . . . .	6
5.0.3	Tài liệu API . . . . .	6
<b>6</b>	<b>Mở rộng</b>	<b>7</b>
6.1	Giới thiệu về Trie . . . . .	7
6.2	Giải thuật tìm kiếm bằng Trie . . . . .	7
6.2.1	Xử lý dữ liệu . . . . .	7
6.2.2	Lưu trữ cây Trie để tối ưu hiệu năng . . . . .	8
6.2.3	Tìm kiếm trên cây Trie . . . . .	8
6.2.4	Quy trình tổng quát . . . . .	8
6.2.5	Kiểm tra tốc độ . . . . .	8

# 1 Tổng quan đề tài

## 1.1 Giới thiệu

Trong thời đại số hóa hiện nay, việc quản lý thông tin tài chính một cách hiệu quả và nhanh chóng đã trở thành nhu cầu thiết yếu đối với cả cá nhân và tổ chức. Đặc biệt, nhu cầu tra cứu thông tin sao kê giao dịch ngân hàng theo các tiêu chí cụ thể như số tiền gửi hoặc nội dung chuyển khoản ngày càng trở nên phổ biến. Tuy nhiên, nhiều hệ thống hiện tại vẫn chưa đáp ứng tốt khả năng lọc dữ liệu linh hoạt và hiển thị thông tin một cách trực quan, thân thiện với người dùng. Đề tài "Thiết kế giao diện (Web, App, CLI) để tra cứu thông tin sao kê theo số tiền gửi và nội dung chuyển khoản" ra đời nhằm cung cấp một giải pháp tối ưu, giúp người dùng dễ dàng truy cập và quản lý dữ liệu giao dịch tài chính của mình một cách nhanh chóng và chính xác.

Đề tài tập trung vào việc xây dựng một hệ thống giao diện hỗ trợ người dùng tra cứu thông tin sao kê giao dịch. Người dùng có thể tìm kiếm thông tin dựa trên các tiêu chí chính như:

Tra cứu theo số tiền gửi: Người dùng có thể nhập giá trị số tiền cụ thể để tìm các giao dịch khớp. Tính năng này đặc biệt hữu ích khi cần kiểm tra các giao dịch lớn hoặc bất thường.

Tra cứu theo nội dung chuyển khoản: Tính năng này hỗ trợ lọc các giao dịch dựa trên từ khóa hoặc cụm từ cụ thể trong nội dung giao dịch, giúp dễ dàng xác minh thông tin hoặc đối chiếu dữ liệu.

Với các tính năng trên, hệ thống này giúp tăng cường khả năng quản lý tài chính cá nhân hoặc doanh nghiệp, hỗ trợ truy vấn nhanh chóng và chính xác các thông tin giao dịch trong tài khoản ngân hàng.

## 1.2 Phạm vi nghiên cứu

Thiết kế giao diện trên nền tảng Web, hiển thị các cấu trúc dữ liệu và giải thuật sử dụng công cụ và thư viện của bên thứ 3 để hiện thực các tính năng chính của hệ thống như:

- Nhập liệu tiêu chí tìm kiếm (số tiền, nội dung chuyển khoản).
- Hiển thị danh sách giao dịch phù hợp với kết quả tìm kiếm.

# 2 Cấu trúc dữ liệu và giải thuật tìm kiếm

## 2.1 Cấu trúc dữ liệu sử dụng

### 2.1.1 Struct Transaction

Struct Transaction được định nghĩa để đại diện cho từng giao dịch. Các trường dữ liệu chính bao gồm:

- `date_time` ( String ): Lưu thời gian giao dịch.
- `trans_no` ( Integer ): Mã định danh giao dịch.
- `credit` ( Unsigned long ): Số tiền gửi.
- `debit` ( Integer ): Số tiền gửi ( ghi nợ )
- `detail` ( String ): Thông tin chi tiết của giao dịch, ví dụ: lý do chuyển tiền hoặc ghi chú.

Struct này phù hợp với mô hình dữ liệu bảng và hỗ trợ các thao tác tìm kiếm, lọc, sắp xếp mà không cần các cấu trúc phức tạp hơn.

### 2.1.2 Vector (std::vector<Transaction>)

Mục đích: Lưu trữ danh sách giao dịch.

Ưu điểm:

- Truy cập ngẫu nhiên với độ phức tạp  $O(1)$ .
- Thao tác thêm, xóa linh hoạt ở cuối vector.

- Tương thích tốt với các thuật toán chuẩn như `std::sort`, `std::lower_bound`, `std::upper_bound`.

**Nhược điểm:** Thao tác thêm hoặc xóa ở giữa có thể tốn kém nếu dữ liệu lớn.

## 2.2 Giải thuật tìm kiếm sử dụng

### 2.2.1 Tìm kiếm theo số tiền gửi (Hàm `creditSearch`)

#### Mô tả giải thuật

Hàm `creditSearch` sử dụng thuật toán tìm kiếm nhị phân để tìm kiếm các giao dịch trong vector (`std::vector<Transaction>`) dựa trên giá trị `credit` nằm trong khoảng `lower_bound_val` đến `upper_bound_val` như sau:

1. Đầu tiên, nhóm sẽ sắp xếp tập dữ liệu theo giá trị tăng dần của `credit` bằng cách sử dụng hàm `std::sort`.
2. Giải thuật sẽ sử dụng `std::lower_bound` để tìm vị trí đầu tiên trong vector mà giá trị `credit` không nhỏ hơn `lower_bound_val`.
3. Sau đó, giải thuật sẽ sử dụng `std::upper_bound` để tìm vị trí đầu tiên trong vector mà giá trị `credit` lớn hơn `upper_bound_val`.
4. Sau khi xác định được hai vị trí, hàm sẽ lặp qua các phần tử nằm trong khoảng này và thêm chúng vào vector `results`.

#### Độ phức tạp của thuật toán

- Thuật toán sắp xếp: `std::sort` sử dụng thuật toán IntroSort và có độ phức tạp là  $O(n \log n)$ .
- Tìm kiếm nhị phân: cả hai hàm `std::lower_bound` và `std::upper_bound` đều có độ phức tạp là  $O(\log n)$ , trong đó  $n$  là số lượng phần tử trong vector.
- Lặp qua các phần tử: việc lặp qua các phần tử giữa hai vị trí tìm được có độ phức tạp là  $O(k)$  trong đó  $k$  là số phần tử nằm trong khoảng tìm kiếm.

Như vậy, độ phức tạp của hàm `creditSearch` là  $O(\log n + k)$ .

#### Ưu điểm:

- Hiệu suất cao: sử dụng tìm kiếm nhị phân giúp giảm thiểu số lượng phép so sánh cần thiết, đặc biệt là khi làm việc với tập dữ liệu lớn.
- Dễ dàng mở rộng: thuật toán có thể dễ dàng điều chỉnh để tìm kiếm theo các tiêu chí khác nhau, chỉ cần thay đổi hàm so sánh.

#### Hạn chế:

- Hiệu suất của chương trình có thể bị giảm nếu số lượng phần tử nằm giữa khoảng tìm kiếm quá lớn.
- Khó duy trì khi dữ liệu thay đổi thường xuyên.

### 2.2.2 Tìm kiếm theo nội dung chuyển khoản (Hàm `detailSearchParallel`)

#### Mô tả giải thuật:

1. Chia nhỏ dữ liệu:

Danh sách giao dịch được chia thành các phần nhỏ (chunks) để phân phối công việc giữa các luồng xử lý. Số lượng phần sẽ được xác định dựa trên số luồng xử lý khả dụng trong hệ thống, tính toán thông qua `std::thread::hardware_concurrency`. Mỗi phần sẽ chứa một số lượng giao dịch tương đương.

2. Xử lý song song:

Mỗi phần giao dịch được phân công cho một luồng riêng biệt thông qua `std::async`, cho phép thực hiện song song mà không làm gián đoạn công việc của các luồng khác. Mỗi luồng thực hiện tìm kiếm trong các trường `detail` của giao dịch để xác định những giao dịch có chứa từ khóa tìm kiếm. Quá trình tìm kiếm

trong mỗi luồng tương tự như trong hàm `detailSearch`, nhưng được tối ưu hóa bằng cách chia nhỏ và xử lý đồng thời.

### 3. Kết hợp kết quả:

Kết quả tìm kiếm từ tất cả các luồng được thu thập và hợp nhất lại thành danh sách kết quả cuối cùng. Sau khi tất cả các luồng hoàn thành công việc, kết quả của mỗi luồng được thu thập từ các đối tượng `std::future`, sau đó ghép nối lại để tạo thành danh sách kết quả đầy đủ.

#### **Độ phức tạp của thuật toán:**

$O(n \cdot m/t)$ , trong đó:

- $n$  là số lượng giao dịch.
- $m$  là độ dài trung bình của trường `detail`
- $t$  là số lượng luồng xử lý

Độ phức tạp này phản ánh sự phân chia công việc đồng đều giữa các luồng và thời gian tìm kiếm trong mỗi luồng.

#### **Ưu điểm:**

- Tăng tốc đáng kể khi làm việc với dữ liệu lớn, tận dụng tốt khả năng tính toán song song của hệ thống.
- Linh hoạt trong việc mở rộng với nhiều luồng xử lý hơn.

#### **Hạn chế:**

- Quản lý luồng và đồng bộ dữ liệu có thể tốn tài nguyên.
- Hiệu suất phụ thuộc vào khả năng xử lý song song của hệ thống.

## 3 Công cụ và công nghệ sử dụng

### 3.1 Công nghệ

#### 3.1.1 Ngôn ngữ C++

Đây là Ngôn ngữ chính được sử dụng cho backend, đặc biệt là trong việc xử lý dữ liệu và thực hiện các phép toán liên quan đến giao dịch.

#### 3.1.2 Thư viện Flask

Framework Python được sử dụng để xây dựng API cho ứng dụng, cho phép giao tiếp giữa frontend và backend.

#### 3.1.3 Next.js

Framework React cho frontend, giúp xây dựng giao diện người dùng với khả năng tối ưu hóa hiệu suất và SEO.

#### 3.1.4 Bootstrap

Thư viện CSS được sử dụng để thiết kế giao diện người dùng, giúp tạo ra các thành phần giao diện đẹp và responsive.

### 3.2 Công cụ

#### 3.2.1 Git

Hệ thống quản lý phiên bản được sử dụng để theo dõi các thay đổi trong mã nguồn và phối hợp làm việc giữa các lập trình viên.

### 3.2.2 Flask-RESTx

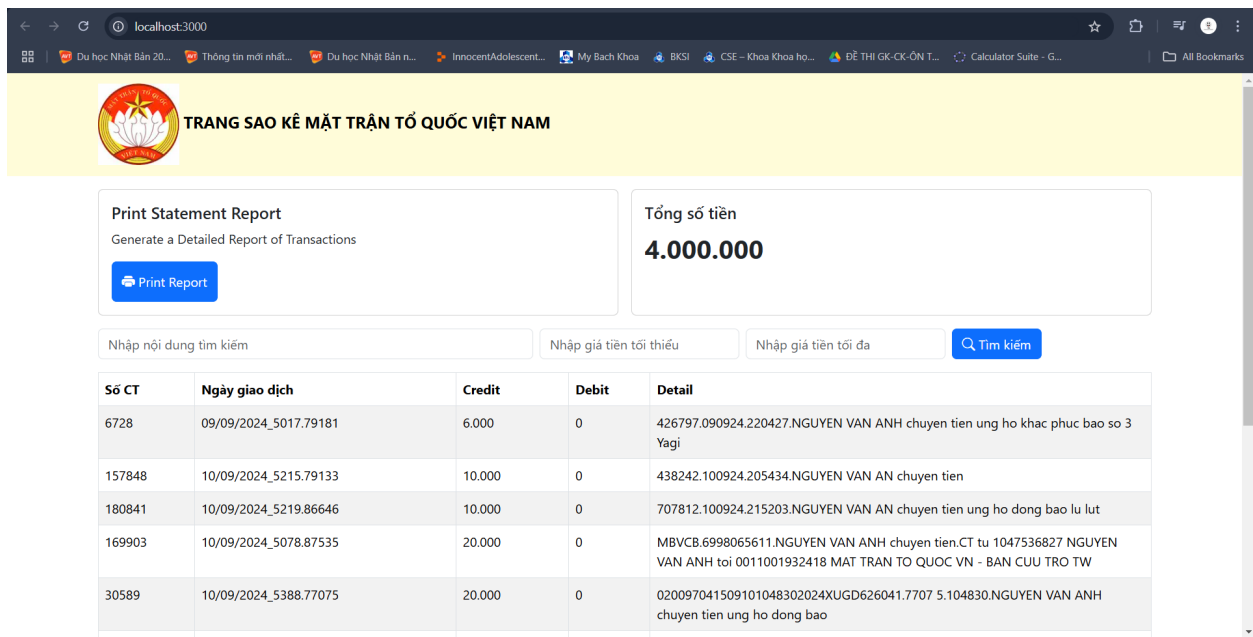
Thư viện mở rộng cho Flask, giúp xây dựng API RESTful một cách dễ dàng và hiệu quả.

### 3.2.3 Swagger UI

Được tạo ra từ Flask-RESTx, là công cụ để kiểm tra API, cho phép gửi yêu cầu và xem phản hồi từ server.

## 4 Thiết kế giao diện người dùng

Dưới đây là giao diện ứng dụng của nhóm:



The screenshot displays a web application interface for "TRANG SAO KÊ MẶT TRẬN TỔ QUỐC VIỆT NAM". It features a "Print Statement Report" section with a "Print Report" button. To the right, the "Tổng số tiền" (Total amount) is displayed as "4.000.000". Below these, there are search filters: "Nhập nội dung tìm kiếm" (Enter search content), "Nhập giá tiền tối thiểu" (Enter minimum price), and "Nhập giá tiền tối đa" (Enter maximum price), followed by a "Tìm kiếm" (Search) button. A table of transactions is shown below the filters, with columns for "Số CT" (Transaction ID), "Ngày giao dịch" (Transaction Date), "Credit", "Debit", and "Detail".

Số CT	Ngày giao dịch	Credit	Debit	Detail
6728	09/09/2024_5017.79181	6.000	0	426797.090924.220427.NGUYEN VAN ANH chuyen tien ung ho khac phuc bao so 3 Yagi
157848	10/09/2024_5215.79133	10.000	0	438242.100924.205434.NGUYEN VAN AN chuyen tien
180841	10/09/2024_5219.86646	10.000	0	707812.100924.215203.NGUYEN VAN AN chuyen tien ung ho dong bao lu lut
169903	10/09/2024_5078.87535	20.000	0	MBVCB.6998065611.NGUYEN VAN ANH chuyen tien.CT tu 1047536827 NGUYEN VAN ANH toi 0011001932418 MAT TRAN TO QUOC VN - BAN CUU TRO TW
30589	10/09/2024_5388.77075	20.000	0	020097041509101048302024XUGD626041.7707 5.104830.NGUYEN VAN ANH chuyen tien ung ho dong bao

Hình 1: Giao diện frontend

Phần đầu trang chứa tiêu đề và logo, làm nổi bật mục đích chính của trang là cung cấp thông tin về các giao dịch của "Mặt trận Tổ quốc Việt Nam". Bên dưới là khu vực hiển thị các nút, bao gồm: nút Print Report dùng để in báo cáo về sao kê (hiện nhóm chưa hiện thực chức năng này), và tổng số tiền thu được.

Bên dưới ta có các form tìm kiếm. Người dùng có thể nhập vào tên của người chuyển, giá trị tối thiểu và giá trị tối đa, và nhấn nút "Tìm kiếm". Thông tin về Số CT, Ngày giao dịch, Số tiền, Debit và Chi tiết sẽ được hiển thị dưới dạng bảng như trên. Với Tanstack Query, dữ liệu được phân trang để có thể tải lên với tốc độ cao hơn. Phương thức search đã được đề cập ở những phần trên và source code sẽ được thêm vào ở phần dưới.

## 5 Kết quả

### 5.0.1 Kiểm thử hiệu năng

Nhóm sẽ dùng thư viện `time` của Python để kiểm tra tốc độ chạy: testcase trên được thực hiện với các parameter:

- nội dung: "NGUYEN THI"
- giá tiền nhỏ nhất: 1000
- giá tiền lớn nhất: 10000000

```

40
41     start = time.time()
42
43     try:
44         # Pass the parameters to the subprocess command
45         result = subprocess.run([os.path.join(os.getcwd(), 'credit'), detail_key, lower_key, upper_key], capture_output=True, text=True)
46         end = time.time()
47         elapsed = end - start
48         print("Total run time:", elapsed)
49         return jsonify({"output": result.stdout})
50     except subprocess.CalledProcessError as e:
51         return jsonify({"error": e.stderr}), 500
52
53 if __name__ == '__main__':
54     port = int(os.environ.get('PORT', 8000))

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```

127.0.0.1 - - [04/Dec/2024 23:47:37] "GET /swaggerui/swagger-ui-bundle.js HTTP/1.1" 200 -
127.0.0.1 - - [04/Dec/2024 23:47:38] "GET /swagger.json HTTP/1.1" 200 -
Total run time: 0.7549948692321777
127.0.0.1 - - [04/Dec/2024 23:48:00] "POST /search HTTP/1.1" 200 -

```

Hình 2: Search

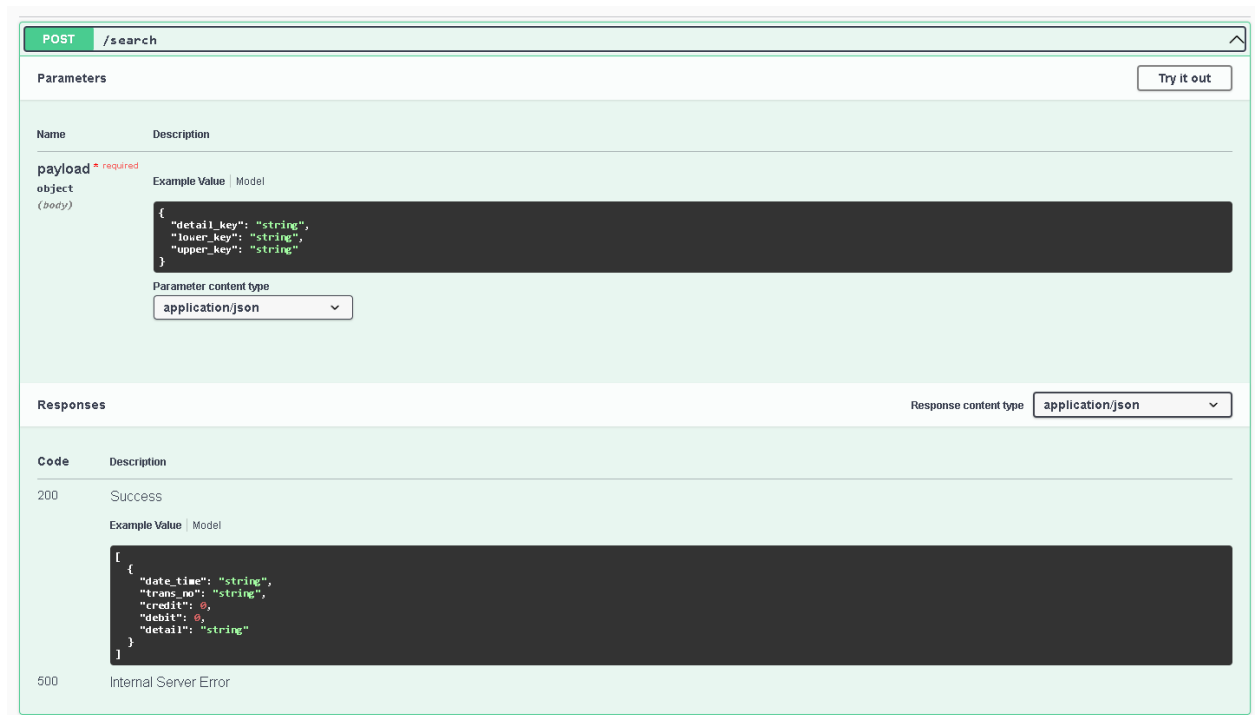
tổng thời gian chạy của chương trình: 0.754 giây

### 5.0.2 Tài nguyên liên quan

- Đường dẫn đến source code: <https://github.com/QuanCters/Advanced-Programming-Assignment.git>
- Đường dẫn đến figma: <https://www.figma.com/design/jsDgJHbiuIwp6GxQOdSSkz/LTNC?node-id=0-1&t=JbkaxMEWh0LezORm-1>

### 5.0.3 Tài liệu API

Nhóm sử dụng Swagger UI với chuẩn OpenAPI 3.0 để trao đổi giữa các thành viên frontend và backend.



Hình 3: Tài liệu trực quan cung cấp bởi Swagger UI

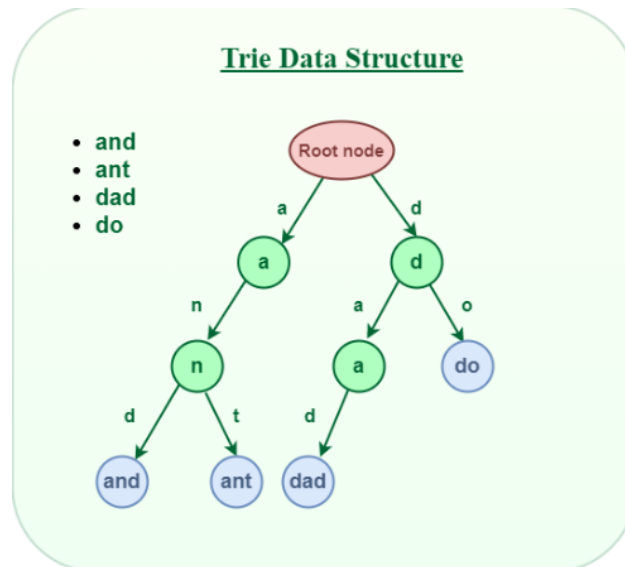
## 6 Mở rộng

Để có thể tối ưu tốc độ tìm kiếm nội dung chuyển khoản(detail). Nhóm đã thử sử dụng cấu trúc dữ liệu Trie để lưu trữ các nội dung chuyển khoản.

### 6.1 Giới thiệu về Trie

Trie, hay còn gọi là Prefix Tree (Cây Tiền Tố), là một cấu trúc dữ liệu dạng cây được sử dụng để lưu trữ và xử lý các chuỗi (strings). Trie đặc biệt hiệu quả trong việc làm việc với các tập dữ liệu lớn gồm các chuỗi, như từ điển, danh sách từ, hoặc các truy vấn tự động hoàn thành.

Mỗi nút trong Trie đại diện cho một ký tự trong chuỗi, và các nhánh từ một nút gốc đến một nút lá tạo thành một chuỗi hoàn chỉnh.



Hình 4: Minh họa Trie

#### Ưu điểm:

Cấu trúc đơn giản, cho phép chèn và tìm kiếm nhanh với độ phức tạp là  $O(k)$  cho chuỗi có độ dài là  $k$ .  
Cho phép xóa với độ phức tạp là  $O(k)$  cho chuỗi có độ dài là  $k$ .  
Phù hợp để lưu trữ những file có dữ liệu lớn, gồm nhiều chuỗi có chung tiền tố.

**Nhược điểm** Không thể thực hiện tìm kiếm gần đúng cho một chuỗi nếu nội dung tìm kiếm không bắt đầu từ chữ cái đầu tiên của chuỗi cần tìm.

Tốn không gian bộ nhớ nếu file gồm nhiều chuỗi với các tiền tố khác nhau, mỗi node sẽ phải lưu một con trỏ node tới rất nhiều chữ cái khác nhau.

### 6.2 Giải thuật tìm kiếm bằng Trie

- Mô tả giải thuật

#### 6.2.1 Xử lý dữ liệu

Đọc file CSV: Tiến hành đọc từng dòng dữ liệu từ file CSV. Mỗi dòng bao gồm các thông tin như  $date_{time}$ ,  $trans_{no}$ ,  $credit$ ,  $debit$ , và  $detail$ . Các dữ liệu  $date_{time}$ ,  $trans_{no}$ ,  $credit$ ,  $debit$  được lưu vào struct record.

Chuyển đổi dữ liệu: Chuỗi detail được chuẩn hóa bằng cách:

Chuyển toàn bộ ký tự thành chữ thường. Loại bỏ các ký tự không phải chữ cái (nếu cần).

Tạo cây Trie: Dữ liệu đã chuẩn hóa được chèn lần lượt vào cây Trie. Mỗi ký tự trong chuỗi detail tương ứng với một node trên cây. Record được lưu tại node cuối của chuỗi.



### 6.2.2 Lưu trữ cây Trie để tối ưu hiệu năng

Vì việc đọc file CSV và xây dựng cây Trie có thể tốn nhiều thời gian, cây Trie sau khi được xây dựng sẽ được serialize và lưu trữ trong file nhị phân *trie\_data.bin*. Lợi ích: Tăng tốc độ khởi chạy chương trình: Ở lần chạy tiếp theo, chỉ cần tải lại cấu trúc Trie từ file Trie.bin, không cần đọc và xử lý lại file CSV. Tiết kiệm tài nguyên: Giảm số lần truy xuất file CSV, đặc biệt khi file lớn.

### 6.2.3 Tìm kiếm trên cây Trie

Tìm kiếm :(hàm `search()`) Với một tiền tố (prefix), chương trình duyệt cây Trie theo các ký tự của tiền tố. Khi kết thúc tiền tố, duyệt tất cả các nhánh con từ node hiện tại để thu thập toàn bộ các bản ghi liên quan.

Hiệu năng:

Giải thuật tìm kiếm trên Trie có thời gian truy xuất trung bình là  $O(m)$ , với  $m$  là độ dài chuỗi tìm kiếm. Trie đặc biệt phù hợp cho các bài toán xử lý chuỗi và tìm kiếm nhanh với dữ liệu lớn.

### 6.2.4 Quy trình tổng quát

Lần chạy đầu tiên:

Đọc file CSV. Chuẩn hóa dữ liệu và xây dựng cây Trie. Serialize cây Trie và lưu vào Trie.bin. Lần chạy sau:

Tải cây Trie từ Trie.bin. Thực hiện tìm kiếm trên cây Trie với từ khóa hoặc tiền tố.

### 6.2.5 Kiểm tra tốc độ

Thư viện `<chrono>` được sử dụng để tính tốc độ chạy của giải thuật tìm kiếm.

1. Tìm kiếm bằng tiền tố "267515.010924.122904.NGUYEN thi mao"

```
Enter keyword to search (or type 'exit' to quit): 267515.010924.122904.NGUYEN thi mao
Search completed in: 2.91e-05 seconds
Results:
01/09/2024_5215.97152 | 1 | 3000 | 0 | 267515.010924.122904.NGUYEN THI MAO Chuyen tien
```

Hình 5: Search

2. Tìm kiếm chính xác "MBVCB.6942540225.HA VAN VINH chuyen tien.CT tu 0731000926523 HA VAN VINH toi 0011001932418 MAT TRAN TO QUOC VN - BAN CUU TRO TW"

```
Enter keyword to search (or type 'exit' to quit): MBVCB.6942540225.HA VAN VINH chuyen tien.CT tu 0731000926523 HA VAN VI
NH toi 0011001932418 MAT TRAN TO QUOC VN - BAN CUU TRO TW
Search completed in: 0.0000192 seconds
Results:
04/09/2024_5242.18550 | 42 | 2000 | 0 | MBVCB.6942540225.HA VAN VINH chuyen tien.CT tu 0731000926523 HA VAN VINH toi 001
1001932418 MAT TRAN TO QUOC VN - BAN CUU TRO TW
```

Hình 6: Search

3. Nhận xét: Nhìn chung quá trình tìm kiếm có tốc độ khá nhanh và trả về kết quả hoàn toàn chính xác. Có thể thấy tốc độ tìm kiếm sẽ bị ảnh hưởng trực tiếp bởi độ dài của chuỗi từ khóa cần tìm. Tuy nhiên như đã nói, nhược điểm của Trie là không thể tìm kiếm gần đúng mà chỉ có thể kiểm theo tiền tố hoặc toàn bộ chuỗi. Đó là nhược điểm khá lớn mà nhóm vẫn chưa tìm được cách khắc phục, vì nhược điểm này làm cho giải thuật tìm kiếm bằng cách sử dụng TRie không còn phù hợp cho mục tiêu của bài tập lớn này nên nhóm quyết định không sử dụng Trie.



## Tài liệu tham khảo

- [1] GeeksforGeeks. *Trie / Insert and Search*. Truy cập vào: 2024-11-20. 2024. URL: <https://www.geeksforgeeks.org/trie-insert-and-search/>.
- [2] Pallets Projects. *Flask Documentation*. Truy cập vào: 2024-11-28. 2024. URL: <https://flask.palletsprojects.com/en/stable/>.
- [3] Swagger. *Swagger UI Documentation*. Truy cập vào: 2024-12-01. 2024. URL: <https://swagger.io/tools/swagger-ui/>.