ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Cấu Trúc Dữ Liệu và Giải Thuật
Học Phần Mở Rộng

NÉN VÀ PHÁT HIỆN CHUỖI BẤT THƯỜNG BẰNG TRIE VÀ PHÂN TÍCH THỐNG KÊ

Giáo viên hướng dẫn: TS. Lê Thành Sách

Lớp: TN01
Nhóm: 13

STT	Họ và tên	MSSV
1	Trần Tấn Phát	2412610
2	Nguyễn Anh Quân	2412899

TP. HỒ CHÍ MINH, 2025

Mã nguồn và demo:

GitHub: https://github.com/quannguyen2412899/SCAD-TSA

Colab: https://colab.research.google.com/github/quannguyen2412899/SCAD-TSA/blob/master/colab/demo.ipynb

Mục lục

CHƯƠNG 1: MỞ ĐẦU

Trong đề tài này, nhóm đã xây dựng một hệ thống sử dụng cấu trúc dữ liệu trie (cây tiền tố) trong việc nén và phân tích dữ liệu chuỗi (log, văn bản, chuỗi sinh học...) nhằm phát hiện chuỗi có hành vi bất thường.

Hệ thống khai thác các kỹ thuật duyệt cây, phân tích nhánh trie, phát hiện bất thường thông qua các tiêu chí về tần suất, độ dài, entropy cục bộ để xác định các chuỗi có độ phổ biến thấp hoặc cấu trúc lạ thường.

Các mục tiêu chính bao gồm:

1.1. Nghiên cứu lý thuyết

- Hiểu và cài đặt cấu trúc Trie.
- Tìm hiểu cách sử dụng Trie để nén dữ liệu chuỗi.
- Phân tích bất thường bằng các đặc trưng thống kê trên Trie.

1.2. Xây dưng hệ thống

- Cài đặt Trie để lưu trữ và nén chuỗi.
- Tính toán các đặc trưng thống kê tại mỗi nút: tần suất, chiều sâu, entropy cục bộ.
- Phát hiện các chuỗi bất thường trên ngưỡng thống kê.

1.3. Trực quan và đánh giá

- Hiển thị cấu trúc Trie và các chuỗi bất thường.
- So sánh mức độ nén trước và sau.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1. Cấu trúc dữ liệu

2.1.1. Tổng quan và định nghĩa

Trie (hay còn gọi là Prefix Tree - Cây tiền tố) là một cấu trúc dữ liệu dạng cây có thứ tự (ordered tree data structure), được sử dụng chuyên biệt để lưu trữ một mảng liên kết các chuỗi ký tự [1]. Khác với Cây tìm kiếm nhị phân (BST) hay Bảng băm (Hash Table), các nút (node) trong Trie không lưu trữ trực tiếp toàn bộ khóa (key). Thay vào đó, vị trí của nút trên cây sẽ xác định nội dung của khóa được liên kết với nó. Về mặt hình thức, một Trie cơ bản thỏa mãn các tính chất sau:

- Nút gốc (Root): Là điểm bắt đầu của cây, thường là một nút rỗng không chứa ký tự.
- Liên kết cạnh: Mỗi cạnh nối từ nút cha sang nút con đại diện cho một ký tự trong bảng chữ cái (ví dụ: ASCII, Unicode).
- Đường dẫn (Path): Một đường dẫn từ nút gốc đến bất kỳ nút u nào trên cây đại diện cho một tiền tố (prefix) chung của các chuỗi đi qua nút đó.
- Nút kết thúc: Các nút đánh dấu sự kết thúc của một chuỗi hoàn chỉnh (thường được định danh bằng cờ isEndOfWord hoặc isLeaf).

2.1.2. Cơ chế nén dữ liệu qua chia sẻ tiền tố

Một trong những ưu điểm lớn nhất của Trie là khả năng chia sẻ tiền tố (prefix sharing). Khi lưu trữ một tập hợp lớn các chuỗi có sự tương đồng về ký tự đầu (các dòng log hệ thống có timestamp giống nhau, các đoạn gen sinh học...), Trie chỉ lưu trữ phần tiền tố chung đó một lần duy nhất.

Cụ thể, nếu hai chuỗi S1 và S2 có chung tiền tố độ dài k, chúng sẽ chia sẻ chung k nút đầu tiên trên cây. Sự phân nhánh chỉ xảy ra tại vị trí ký tự thứ k+1 khác biệt. Cơ chế này giúp Trie thực hiện việc nén dữ liệu một cách tự nhiên, giảm thiểu đáng kể không gian lưu trữ dư thừa so với việc lưu trữ danh sách chuỗi độc lập, đặc biệt hiệu quả với các bộ từ điển lớn hoặc dữ liệu có tính lặp lại cao.

Độ nén được tính theo tỉ lệ giữa tổng ký tự của các từ phân biệt được lưu trong trie với tổng số nút của trie:

$$\text{compression rate} = \frac{\sum_{word \in \text{all unique words}} \text{len(word)}}{\text{total nodes}}$$

2.2. Các tiêu chí xác định bất thường

Hệ thống được thiết kế để đánh giá bất thường theo 3 tiêu chí: tần suất, độ dài và entropy cục bộ (của một nút trên trie). Trên thực tế, có nhiều cách để định nghĩa thế nào là một chuỗi bất thường, ví dụ, có thể nói rằng một chuỗi hiếm khi xuất hiện trong một văn bản là bất thường, nhưng trong một vài ngữ cảnh khác, một chuỗi xuất hiện quá nhiều lần cũng có thể được xem là bất thường. Vì vậy, trong phạm vi đề tài này quy ước như sau:

- Dựa trên tần suất: một chuỗi có tần suất xuất hiện càng ít thì càng bất thường
- Dựa trên độ dài: một chuỗi có độ dài có tần suất càng thấp thì càng bất thường. Nghĩa là, nếu số chuỗi có độ dài a ít hơn số chuỗi có độ dài b thì tất cả các chuỗi có độ dài bằng a sẽ bất thường hơn số tất cả chuỗi có độ dài bằng b.
- Dựa trên entropy:

  Định nghĩa: Entropy thông tin là một thước đo về tính ngẫu nhiên của một sự kiện, entropy càng cao, sự kiện càng khó đoán.

  Entropy cục bộ tại mỗi nút được tính toán dựa trên phân phối xác suất của các nhánh con và cả xác suất chuỗi kết thúc (isEnd) tại nút đó. Giá trị entropy cục bộ của H(u) của node u được xác định theo công thức Shannon [2] và được mở rộng như sau:

$$H(u) = -\left(\sum_{v \in \text{children}(u)} p(v) \log_2 p(v) \right) - p_{\text{end}} \log_2 p_{\text{end}}$$

  Đường đi từ root đến node đang xét entropy sẽ tạo thành một tiền tố. Nếu node này có entropy càng cao, tiền tố tương ứng với node này sẽ càng bất thường.

2.3. Phương pháp thống kê và phát hiện bất thường

Các bất thường sẽ được xác định dựa trên ngưỡng của từng tiêu chí, và để phù hợp với các tập dữ liệu có kích thước khác nhau, ngưỡng sẽ không cố định mà được xác định dựa bằng phương pháp **bách phân vị** [3].

Định nghĩa: Bách phân vị thứ k (P_k) của một tập dữ liệu là giá trị mà tại đó k% số quan sát trong tập dữ liệu có giá trị nhỏ hơn hoặc bằng nó.

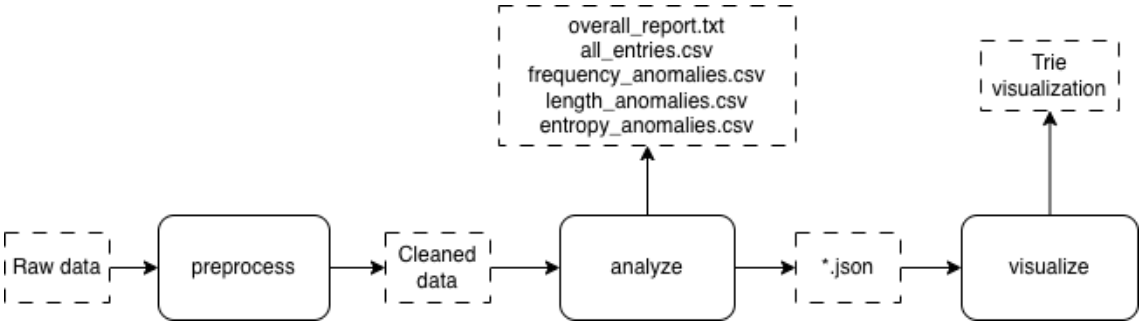
Việc sử dụng bách phân vị cho phép hệ thống thiết lập các ngưỡng "động" tự động thích nghi với quy mô và đặc tính phân phối của từng file dữ liệu đầu vào mà không cần tham số cứng.

Các bất thường theo từng tiêu chí sẽ được xác định dựa trên ngưỡng của từng tiêu chí, đối với tần suất và độ dài, các bất thường là các chuỗi có tần suất xuất hiện hoặc tần suất độ dài thấp hơn ngưỡng, đối với entropy, một tiền tố có entropy cao hơn ngưỡng thì bị coi là bất thường.

CHƯƠNG 3: XÂY DỰNG HỆ THỐNG

3.1. Tổng quan kiến trúc

Toàn bộ mã nguồn đã được upload trên Github repository đính kèm ở phần Mở đầu. Về kiến trúc, hệ thống được thiết kế dưới dạng một chuỗi lệnh (pipeline) gồm 3 module được điều phối bởi executable main_pipeline.



3.2. Hiện thực Trie

Để phục vụ việc lưu trữ chuỗi và thông tin thống kê chi tiết cho module phân tích, nhóm đã hiện thực lớp StatTrie (Statistical Trie).

3.2.1. Cấu trúc Nút (Node)

Đơn vị cơ bản của cây là struct Node, được thiết kế để lưu trữ thông tin tại mỗi bước chuyển đổi ký tự:

```
struct Node {
    std::unordered_map<char, Node*> children;
    unsigned count;
    bool isEnd;

    Node();
    unsigned countEnd() const;
};
```

- Quản lý liên kết (children): Việc sử dụng std::map giúp cây linh hoạt với các bộ ký tự mở rộng (thay vì mảng tĩnh 26 phần tử), đồng thời tối ưu hóa bộ nhớ tại các nút thưa (sparse nodes).

- Thống kê cục bộ:
    + count (unsigned): Đếm số lượng chuỗi đi qua nút này. Giá trị này đại diện cho tần suất của tiền tố (prefix frequency), là cơ sở để tính xác suất chuyển nhánh $P(v)$.
    + countEnd() (unsigned): Đếm số lượng chuỗi kết thúc chính xác tại nút này.
    + isEnd (bool): Cờ đánh dấu điểm kết thúc của một từ hoàn chỉnh.

3.2.2. Quản lý Thống kê Toàn cục

Lớp StatTrie duy trì các biến đếm toàn cục để cung cấp thông tin tổng quan về tập dữ liệu mà không cần duyệt lại cây.

- totalInsertedWords: Tổng số chuỗi đã chèn (bao gồm cả trùng lặp).
- totalUniqueWords: Tổng số chuỗi duy nhất.
- totalNodes: Tổng số nút trong cây (đo lường kích thước bộ nhớ).
- totalUniqueWordChar: Tổng số ký tự của các từ duy nhất (dùng để tính độ nén).

3.2.3. Phương thức chèn và cập nhật

Phương thức insert(const std::string &word) thực hiện hai nhiệm vụ song song: xây dựng cấu trúc và cập nhật thống kê thời gian thực.
Quy trình: Duyệt qua từng ký tự của chuỗi đầu vào. Tại mỗi bước:

- Nếu nhánh con chưa tồn tại, tạo nút mới và tăng biến totalNodes cùng totalUniqueWordChar.
    + Nếu nhánh con chưa tồn tại, tạo nút mới và tăng biến totalNodes cùng totalUniqueWordChar.
    + Tăng biến count tại nút hiện tại (ghi nhận tần suất đi qua).
- Kết thúc: Tại nút cuối cùng, tăng biến countEnd, đặt cờ isEnd = true. Nếu đây là từ mới (lần đầu xuất hiện), tăng totalUniqueWords.

3.2.4. Phương thức duyệt cây

Để hỗ trợ module Phân tích tính toán Entropy và thu thập dữ liệu, StatTrie cung cấp cơ chế duyệt cây linh hoạt thông qua hàm traverse:

- Cơ chế callback: Hàm nhận vào một std::function (callback). Điều này cho phép tách biệt logic duyệt cây (nằm trong StatTrie) và logic xử lý nghiệp vụ (nằm trong Analysis).
- Thuật toán: được overload thành 2 phiên bản
  + Duyệt tiền thứ tự toàn bộ cây.
  + Duyệt theo chiều sâu tương ứng với một tiền tố.

Tại mỗi nút được ghé thăm, hàm callback được gọi, cung cấp quyền truy cập trực tiếp vào con trỏ Node hiện tại và chuỗi tiền tố tương ứng. Điều này cho phép module bên ngoài tính toán các chỉ số phức tạp (như Entropy cục bộ) dựa trên dữ liệu thô của nút đó.

Lưu ý: Mặc dù các phương thức như remove hay contains cũng được hiện thực đầy đủ trong mã nguồn để đảm bảo tính trọn vẹn của cấu trúc dữ liệu, chúng không được sử dụng trong quy trình phát hiện bất thường chính nên không được đề cập chi tiết tại đây.

3.3. Các module chức năng

3.3.1. Tiền xử lí (module preprocess)

Module Tiền xử lí chịu trách nhiệm chuyển đổi dữ liệu thô đầu vào thành một tập chuỗi đã chuẩn hoá, sạch và sẵn sàng để đưa vào các bước phân tích tiếp theo (xây dựng Trie, tính thống kê, phát hiện bất thường). Module này được triển khai thành hai thành phần chính: chương trình điều khiển (preprocess.cpp) đảm nhiệm giao diện dòng lệnh và phối hợp luồng xử lý, và lớp Preprocessor (định nghĩa trong Preprocessor.cpp / Preprocessor.h) chứa toàn bộ logic làm sạch chuỗi.

Mục tiêu chính:

- Lọc/chiết xuất chuỗi theo biểu thức chính quy (regex) khi người dùng yêu cầu.

- Chuẩn hoá kí tự (lowercase), loại bỏ kí tự không mong muốn, phân tách dựa trên dấu phân cách do người dùng cung cấp.

- Chuẩn hoá khoảng trắng (gộp nhiều space thành một, loại space đầu/cuối).

- Xuất kết quả ra file đầu ra theo dòng, đồng thời trả về danh sách các chuỗi đã thu thập để các module khác sử dụng.

Quy trình thực hiện gồm các bước chính sau:

Bước 1: Tiếp nhận và Cấu hình tham số

Khi hệ thống khởi chạy, module tiền xử lý phân tích các tham số dòng lệnh để xác định cách xử lý dữ liệu phù hợp. Các tùy chọn này quyết định toàn bộ hành vi của pipeline:

- Chế độ lọc Regex (–regex): Cho phép chỉ định một mẫu biểu thức chính quy. Tuy chỉ là một tham số nhỏ, nhưng nó thay đổi toàn bộ chiến lược xử lý: hệ thống không còn xem dữ liệu như văn bản thô, mà tập trung truy xuất thông tin có cấu trúc (IP, timestamp, token sinh học. . .). Điều này giúp giảm mạnh nhiễu ở các dataset phức tạp.

- Loại bỏ ký tự nhiễu (–ignore): Người dùng có thể định nghĩa danh sách ký tự "không có ý nghĩa thống kê", ví dụ dấu câu hoặc ký hiệu dư thừa. Việc loại bỏ ngay từ đầu tránh được việc tạo ra các chuỗi vô ích trong Trie.

- Phân tách chuỗi (–delim): Các ký tự phân cách giúp chia một dòng thành nhiều đơn vị nhỏ hơn. Cách phân tách này đặc biệt quan trọng với log system khi một dòng có nhiều sự kiện ghép lại.

- Chuẩn hóa chữ hoa/thường: Toàn bộ chuỗi được chuyển về lowercase để đảm bảo tính đồng nhất giữa các nguồn dữ liệu, tránh trường hợp "Error", "ERROR", "error" bị coi là ba chuỗi khác nhau.

Bước 2: Xử lý và Chuẩn hóa dữ liệu

Sau khi thiết lập cấu hình, hệ thống tiến hành chuẩn hóa dữ liệu dựa trên hai nhánh xử lý tương ứng. Mục tiêu của bước này là biến dữ liệu thô — vốn có thể chứa nhiều ký tự nhiễu, định dạng không nhất quán hoặc cấu trúc phức tạp — thành các chuỗi sạch, đồng nhất và sẵn sàng để đưa vào Trie.

Phương pháp 1: Trích xuất bằng Regex

Khi bật cờ –regex, hệ thống chuyển sang chế độ "lọc mục tiêu", chỉ tập trung

vào những phần tử có giá trị phân tích cao. Mỗi dòng dữ liệu được quét bằng std::regex_search, từ đó:

- Những đoạn khớp với mẫu Regex sẽ được tách ra như các đơn vị thông tin độc lập.
- Các nhóm khớp (capture groups) cho phép thu thập nhiều mảnh dữ liệu từ cùng một dòng, giảm tải cho phần hậu xử lý.
- Cách làm này đặc biệt hữu ích với các tập dữ liệu có định dạng phức tạp nhưng ổn định, ví dụ: log hệ thống, bản ghi timestamp,...

Nhờ đó, toàn bộ phần "nhiễu cấu trúc" được loại bỏ ngay từ đầu, giúp dataset kết quả gọn, chính xác và tối ưu cho bước xây dựng Trie.

Phương pháp 2: Làm sạch và Cắt chuỗi (Standard Cleaning) Nếu không sử dụng Regex, hệ thống áp dụng phương pháp xử lý ký tự-thuần túy, phù hợp cho dữ liệu không có định dạng rõ ràng. Quá trình gồm các bước:

- Loại bỏ ký tự nhiễu: Mọi ký tự nằm trong danh sách ignore sẽ bị bỏ qua trong lúc duyệt. Điều này giúp loại sạch những ký tự hình thức mà không ảnh hưởng đến nội dung thật sự.
- Tách chuỗi bằng delim: Khi gặp ký tự phân cách, hệ thống xem đây như một "điểm ngắt logic". Điều này cho phép chia một dòng văn bản dài - vốn có thể chứa nhiều thông tin - thành các sequence nhỏ, độc lập, phù hợp với cấu trúc dữ liệu dạng cây Trie.
- Chuẩn hóa khoảng trắng: Không chỉ xóa khoảng trắng dư thừa ở đầu và cuối, hệ thống còn nén các chuỗi khoảng trắng liên tiếp thành một khoảng trắng duy nhất. Việc này tránh được tình trạng sinh ra các chuỗi khác nhau chỉ vì khác số lượng dấu cách.

Phương pháp này làm sạch dữ liệu một cách tự nhiên và ổn định, biến văn bản thô thành các chuỗi có cấu trúc nhẹ, mạch lạc, nhất quán theo từng ký tự.

Bước 3: Xuất dữ liệu chuẩn hóa
Sau khi hoàn tất quá trình lọc và chuẩn hóa, hệ thống chuyển sang giai đoạn đưa dữ liệu sạch ra file. Đây là bước cuối trong pipeline tiền xử lý trước khi dữ liệu được chuyển sang module Analyze và xây dựng Trie.

- Ghi tuần tự từng sequence: Mỗi chuỗi đã được xử lý sẽ được ghi xuống file dưới dạng một dòng duy nhất. Việc xuất tuần tự giúp đảm bảo dữ liệu đầu ra đơn giản, dễ đọc và phù hợp với quá trình nạp từng dòng vào Trie.

- Phân biệt hai chế độ xuất: Ở chế độ Regex, hệ thống bỏ qua toàn bộ cơ chế làm sạch ký tự (ignore, delim, chuẩn hóa khoảng trắng) và chỉ xuất các đoạn khớp đúng như chúng xuất hiện trong dữ liệu gốc.

3.3.2 Phân tích (module analyze)

Module Phân tích (bin/analyze) chịu trách nhiệm xây dựng cấu trúc Trie từ dữ liệu đã tiền xử lý, tính toán các đặc trưng thống kê và phát hiện các chuỗi bất thường dựa trên các ngưỡng động. Quy trình thực hiện bao gồm các bước chính sau:

Bước 1: Duyệt cây và thu thập thống kê
Quá trình này được thực hiện thông qua duyệt tiền thứ tự toàn bộ cây Trie. Tại mỗi nút u được ghé thăm, hệ thống thực hiện

- Tính tần suất xuất hiện của chuỗi: frequency = u.countEnd()

- Tính entropy cục bộ tại node u theo công thức ở phần 2.2, với:

$$p(v) = \frac{v.\text{count}}{u.\text{count}} \quad : \text{ xác suất đi vào nhánh con } v$$

$$p(\text{end}) = \frac{u.\text{countEnd}()}{u.\text{count}} \quad : \text{ xác suất chuỗi kết thúc tại } u$$

- Hệ thống xây dựng một bảng phân phối tần suất cho độ dài chuỗi lenFreq - một mapping từ một độ dài đến số lượng chuỗi có độ dài đó. Như đã bàn ở phần trước, nếu số lượng chuỗi mang độ dài này càng ít, độ dài đó sẽ được coi là càng bất thường.

- Lưu trữ dữ liệu trung gian: Các thông tin thu thập được (bao gồm chuỗi, tần suất, độ dài, entropy) được lưu vào danh sách allEntries để phục vụ bước tính toán ngưỡng.

Bước 2: Tính toán ngưỡng
Hệ thống sử dụng phương pháp Bách phân vị (Percentile) để xác định ngưỡng

động, giúp thuật toán thích nghi với các tập dữ liệu có quy mô và phân phối khác nhau.

- Sắp xếp: Các danh sách giá trị tần suất chuỗi, giá trị entropy và tần suất độ dài được sắp xếp tăng dần.

- Xác định ngưỡng: Ngưỡng tương ứng với từng loại bất thường sẽ được chọn tại vị trí index tương ứng với tham số bách phân vị đầu vào (k%)

$$Index = \frac{k}{100 * N}$$

Sử dụng bách phân vị thấp (mặc định P_5) để xác định các chuỗi hiếm gặp.
Sử dụng bách phân vị thấp (mặc định P_5) trên tập dữ liệu phân phối độ dài để xác định các độ dài hiếm.
Sử dụng bách phân vị cao (mặc định P_{95}) để xác định các nút có độ hỗn loạn cao bất thường.

Bước 3: Phát hiện và phân loại bất thường Dựa vào ngưỡng xác định được từ bước 2, chương trình tiến hành duyệt lại danh sách allEntries. Nếu một chuỗi có:

- Tần suất xuất hiện dưới ngưỡng.

- Tần suất độ dài dưới ngưỡng.

- Entropy vượt ngưỡng tương.

Thì sẽ được trích xuất và phân loại vào các danh sách riêng biệt (freqAnomalies, lenAnomalies, entropyAnomalies), sắp xếp theo mức độ bất thường và xuất ra định dạng CSV/JSON phục vụ báo cáo và trực quan hóa.

Bước 4: Hỗ trợ trực quan hoá (định dạng JSON)
Đối với cây Trie có kích thước lớn, việc vẽ toàn bộ cây trở nên bất khả thi về mặt hiệu năng và khó quan sát, vì vậy xuất hiện nhu cầu thu gọn cây. Quy trình thu gọn cây được thực hiện như sau:

- Đánh dấu Bất thường: chương trình tập hợp tất cả các địa chỉ (con trỏ) của các nút bị phát hiện là bất thường vào một tập hợp đánh dấu (unordered_set<Node*>).

- Duyệt cây Trie từ gốc, tại mỗi nút, thuật toán kiểm tra xem nhánh con của nó có chứa bất kỳ nút được đánh dấu nào không.

+ Một nút được giữ lại trong cây hiển thị nếu chính nó là một nút bất thường hoặc nằm trên đường đi dẫn đến một nút bất thường.
+ Các nút không thuộc trường hợp trên sẽ bị bỏ qua, không đưa vào file JSON đầu ra.

- Định dạng JSON: một file JSON có cấu trúc phân cấp.

Mỗi đối tượng JSON đại diện cho một nút và chứa các siêu dữ liệu cần thiết cho việc vẽ biểu đồ:

- Children: tập hợp các nút con
- Color: màu sắc hiển thị của nút, nút bất thường sẽ có màu đỏ
- Count: số đếm trên nút
- Id: mã định danh duy nhất của một nút hiển thị
- isEnd: flag xác định một nút kết thúc của một chuỗi
- Label: ký tự hiển thị trên nút (và trên cạnh tới nút đó)

```
"root": {
  "children": {
    "H": {
      "children": {},
      "color": "red",
      "count": 1,
      "id": 1,
      "isEnd": true,
      "label": "H"
    }
  },
  "color": "black",
  "count": 0,
  "id": 0,
  "isEnd": false,
  "label": "root"
}
```

3.3.3 Trực quan hoá (module visualize)

Module này được thiết kế kết hợp giữa C++ và Python để tận dụng thư viện đồ họa mạnh mẽ Graphviz.
Về cơ chế tích hợp: Do C++ không có các thư viện vẽ đồ thị linh hoạt như Python, hệ thống sử dụng một chương trình C++ (src/visualize.cpp) đóng vai trò là lớp vỏ (wrapper).

- Chức năng: wrapper tiếp nhận các tham số dòng lệnh từ pipeline chính, xây dựng câu lệnh hệ thống (std::string command) và sử dụng hàm std::system() để gọi trình thông dịch Python thực thi script vẽ hình.
- Lợi ích: Đảm bảo tính nhất quán của giao diện dòng lệnh toàn hệ thống, đồng thời tận dụng được sức mạnh xử lý đồ họa của Python.

Về quy trình dựng cây:

- Bước 1: Đọc file JSON đầu vào
- Bước 2: Sử dụng thuật toán duyệt tiền thứ tự để tái tạo cấu trúc cây, file JSON đã chứa đầy đủ thông tin về màu sắc, ID, label... script python chỉ việc dựa vào đó và đặt cấu hình hiển thị.
- Bước 3: Trực quan hoá - đối tượng đồ thị được biên dịch thành file ảnh thông qua Graphviz Engine.

CHƯƠNG 4: CÀI ĐẶT VÀ HƯỚNG DẪN SỬ DỤNG

Chi tiết hướng dẫn cài đặt và chạy chương trình xem tại Github/Colab.

4.1. Yêu cầu hệ thống và phụ thuộc

- C++
  + Compiler: GNU C++ Compiler (g++), hỗ trợ C++17 trở lên.
  + Thư viện Bên ngoài: nlohmann/json [4]
- Python
  + Python: Python 3.9.
  + Thư viện Python: graphviz. Cài đặt: pip install graphviz.
- Graphviz Engine.
  + Trên Debian/Ubuntu: sudo apt-get install graphviz
  + Trên macOS (Homebrew): brew instal graphviz
  + Hoặc tải trực tiếp từ website: https://graphviz.org/

4.2. Biên dịch

```
mkdir -p bin
g++ -std=c++17 -I./include src/preprocess.cpp src/Preprocessor.cpp -o bin/preprocess
g++ -std=c++17 -I./include src/analyze.cpp src/Analysis.cpp src/StatTrie.cpp -o bin/analyze
g++ -std=c++17 -I./include src/visualize.cpp -o bin/visualize
g++ -std=c++17 -I./include src/main_pipeline.cpp -o main_pipeline
```

4.3. Chạy chương trình

```
    ./main_pipeline <input_file> <output_dir> [flags]
```

CHƯƠNG 5: THỰC NGHIỆM VÀ KẾT QUẢ

5.1. Dữ liệu

Kết quả demo trên colab được thực hiện trên một tập dữ liệu giả lập và một tập dữ liệu thực tế.

- Dữ liệu giả lập: Được sinh ngẫu nhiên để mô phỏng log máy chủ web bị tấn công.
- Dữ liệu thực tế: Sử dụng các file log hệ thống lấy từ môi trường macOS.

5.3. Một số kết quả thu được

5.3.1. File báo cáo tổng quát

```
==================== TRIE ANALYSIS REPORT ====================
----------------------- Trie statistics ----------------------
- Total inserted words: 2209
- Total unique words: 2040
- Total unique-word characters: 187565
- Total nodes: 30200
- Compressed rate (total unique-word characters / total nodes): 6.21076
---------------------- Extremum statistics ---------------------
Word frequency:
- Max frequency: 66
- Min frequency: 1
Word length (depth):
- Max length: 98
- Min length: 9
Entropy (local node entropy):
- Max entropy: 3.96814
- Min entropy: 0
------------------- ANOMALIES DETECTION --------------------
------------------------- Thresholds --------------------------
- Word frequency threshold (0.1% lower percentile): 1
- Length frequency threshold (0.1% lower percentile): 1
```

```
- Entropy threshold (99.9% upper percentile): 3.96814
------------------- Anomalies: frequency-based -------------------
accessoryd[45008], frequency = 1
ampartworkagent[50122], frequency = 1
awdd[209], frequency = 1
...
There are 1980 frequency-based anomalies
Accounted for 89.6333% of the processed text
------------------ Anomalies: length-frequency-based ------------------
ampartworkagent[50122], length = 22, length frequency = 1, frequency = 1
awdd[209], length = 9, length frequency = 1, frequency = 1
com.apple.bkagentservice[49476], length = 31, length frequency = 1, frequency
...
There are 15 length-frequency-based anomalies
Accounted for 0.67904% of the processed text
------------------ Anomalies: entropy-based ------------------
com.apple.xpc.launchd[1] (com.apple.webkit.networking., entropy = 3.96814, fr
There are 1 entropy-based anomalies
Accounted for 16.6139% of the processed text
======================== END OF REPORT ============================
```

5.3.2. Bảng biểu

String	Kind	Frequency	Length	Length frequency	Entropy	Rate	Anomaly
accessoryd[45008]	word	1	17	14	0	0.000453	frequency
ampartworkagent[50122]	word	1	22	1	0	0.000453	frequency/length
awdd[209]	word	1	9	1	0	0.000453	frequency/length
com.apple.appkit.xpc...	word	1	51	4	0	0.000453	frequency
com.apple.appkit.xpc...	word	1	51	4	0	0.000453	frequency

5.4. Đánh giá

Qua thực nghiệm trên các bộ dữ liệu thực tế, hệ thống ghi nhận một số điểm đáng chú ý:

- Tỷ lệ chuỗi được đánh dấu là bất thường có thể chiếm từ 50% đến 90% tổng số chuỗi duy nhất do đa số chuỗi đều chỉ xuất hiện từ 1-2 lần trong dữ liệu được làm sạch. Như vậy kết quả của chương trình phụ thuộc nhiều vào phân phối của dữ liệu.

- Hiệu năng của module tiền xử lý phụ thuộc mạnh vào biểu thức chính quy, việc lọc bằng regex nếu sử dụng không hợp lý có thể dẫn đến tốc độ xử lý chậm.

- Đồng thời việc trực quan hóa cây trie vẫn còn hạn chế về mặt thời gian cũng như mức độ trực quan. Khi ở tập dữ liệu lớn từ vài nghìn node trở lên, thời gian chạy sẽ không còn tối ưu. Graphviz vẽ cây theo tầng, đồng thời cũng phải scale hình ảnh, do đó, với cây trie lớn hoặc có nhiều chuỗi dài thì hình ảnh sẽ trở nên khó quan sát.

CHƯƠNG 6: TỔNG KẾT

6.1. Tổng kết các kết quả đạt được

Đề tài đã thiết kế và hiện thực thành công hệ thống SCAD-TSA. Tổng kết các kết quả đã đạt được:

- Hiện thực và ứng dụng hiệu quả cấu trúc Trie: cấu trúc Trie được triển khai thành công, đạt tỉ lệ nén trung bình 2,5–3,5 lần (có thể lên đến >6 lần đối với bộ dữ liệu có tính lặp lại cao).
- Phương pháp phân tích thống kê: thay vì sử dụng các phương pháp thống kê tham số truyền thống, đề tài đã áp dụng thành công phương pháp bách phân vị (Percentile) để xác định ngưỡng bất thường. Cách tiếp cận này không chỉ đơn giản mà còn giúp cho ngưỡng bất thường "thích nghi" với kích thước văn bản.
- Hệ thống pipeline hoàn chỉnh: xây dựng được một quy trình xử lý khép kín từ Tiền xử lý, Phân tích (bằng C++) đến Trực quan hóa (bằng Python / Graphviz). Cơ chế "cắt tỉa cây" trong module trực quan hóa là một giải pháp giúp quan sát và phân tích các cấu trúc dữ liệu lớn.

6.1. Hạn chế

Trong quá trình thiết kế và xây dựng chương trình, nhóm cũng không thể tránh khỏi sai sót. Một vài hạn chế mà nhóm rút ra trong quá trình thực nghiệm:

- Module tiền xử lý khi sử dụng chế độ lọc bằng regex vẫn chưa được tối ưu.
- Giới hạn của trực quan hóa tĩnh: Hình ảnh xuất ra từ Graphviz trở nên quá khổ và khó quan sát kể cả khi đã hình ảnh đã được thu gọn khi số lượng điểm bất thường hoặc độ sâu của chuỗi tăng cao.
- Việc sử dụng std::map tại mỗi nút Trie tiêu tốn lượng RAM đáng kể cho việc quản lý con trỏ.

TÀI LIỆU THAM KHẢO

[1] C. A. Shaffer, *Data Structures and Algorithm Analysis in C++*, 3rd ed. (Edition 3.2), Dover Publications, 2011, Sec. 13.1.

[2] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[3] J. L. Devore, *Probability and Statistics for Engineering and the Sciences*, 9th ed. Cengage Learning, 2015.

[4] N. Lohmann, "JSON for Modern C++," 2023. [Online]. Available: https://github.com/nlohmann/json.