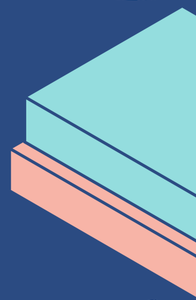
HIỆN THỰC

WEB TRA CỨU SAO KÊ

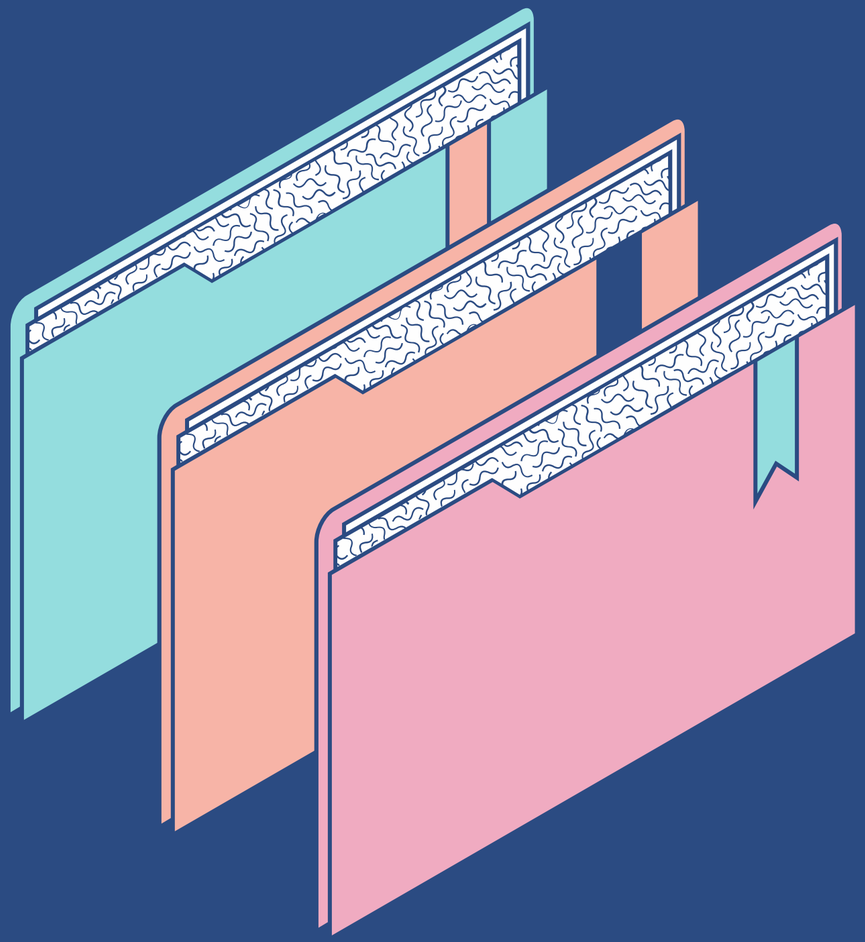


BTL Lập trình nâng cao - CO2039

ID nhóm - 508

THÀNH VIÊN - Nhóm 508

1. VŨ NGỌC THUẬN - 2112394
2. HOÀNG VĂN HẢI - 2111134
3. PHAN THANH BÌNH - 2210332
4. LÊ NGỌC TÂM - 2213018
5. NGUYỄN HỒNG MINH - 2212059



CHƯƠNG TRÌNH

CÁC CHỦ ĐỀ CHÍNH ĐƯỢC THẢO LUẬN
TRONG BẢN THUYẾT TRÌNH NÀY

1. Phân tích yêu cầu
2. Cơ sở lý thuyết
3. Thiết kế hệ thống
4. Hiện thực website
5. Demo

1. Phân tích yêu cầu

1.1. Tổng quan yêu cầu

- Xây dựng được một website cho phép người dùng tra cứu dữ liệu sao kê từ một nguồn dữ liệu nào đó.

- Giúp người dùng tra cứu dữ liệu sao kê một cách nhanh chóng khi nguồn dữ liệu cần tra cứu là quá lớn

1. Phân tích yêu cầu

1.2. Phân tích bài toán

- Bài toán mà website được hiện thực cần giải quyết là tra cứu dữ liệu sao kê.
- Bản chất là bài toán tìm kiếm dữ liệu (search data problem)
- Dữ liệu cần tra cứu là sao kê ngân hàng (các giao dịch ngân hàng).

	A	B	C	D	E	F	G	H	I	J	K	L
1	date_time	trans_no	credit	debit	detail							
2	01/09/2024	1	3000	0	267515.010924.122904.NGUYEN THI MAO Chuyen tien							
3	01/09/2024	2	10000	0	018806.010924.213139.chuc mung ngay 29 nuoc cong hoa xa hoi chu nghia Viet Nam							
4	02/09/2024	3	10000	0	055464.020924.064157.Ung Ho Nha Nuoc Viet Nam Va Giup do Nguoi dan (by TPBank ChatPay							
5	02/09/2024	4	10000	0	MBVCB.6924605040.gia dinh Dung Thuy Giang chuc to quoc khoe manh, binh an, hanh phuc, th							
6	02/09/2024	5	50000	0	MBVCB.6925071164.mung ngay quoc khanh.CT tu 1028808193 toi 0011001932418 Uy Ban Tru							
7	02/09/2024	6	2000	0	524322.020924.175952.2091945							
8	02/09/2024	7	500000	0	020097041509022027252024Ppe432198.64042. 202714.NGUYEN THI LAN HANH Quyen gop							
9	03/09/2024	8	10000	0	881384.030924.070324.Ung Ho Nha Nuoc Viet Nam Va Giup do Nguoi dan							
10	03/09/2024	9	200000	0	120167.030924.100642.NGUYEN HOAI NAM ung ho							
11	03/09/2024	10	300000	0	069690.030924.111352.Ung ho dong bao Viet Nam FT24248787369079							
12	03/09/2024	11	370000	0	524035.030924.111445.IBFT TRAN THI MY PHUONG chuyen tien							
13	03/09/2024	12	100000	0	160707.030924.112000.NGUYEN HOAI PHUONG CK BCT TW							

1. Phân tích yêu cầu

1.2. Phân tích bài toán

- Bài toán tìm kiếm dữ liệu có thể chia thành hai bài toán nhỏ hơn:
  - Tìm kiếm dữ liệu/nội dung cụ thể - thường áp dụng với tất cả các dạng dữ liệu.
  - Tìm kiếm dữ liệu/nội dung theo khoảng giá trị - thường áp dụng với các nội dung có dữ liệu dạng số hoặc ngày giờ.

1. Phân tích yêu cầu

1.3. Các mục tiêu đặt ra

- Website hiện thực có giao diện giúp người dùng tra cứu thông tin sao kê:
  - Tra cứu thông tin theo nội dung cụ thể: tra cứu giao dịch theo ngày giao dịch, số tiền hoặc theo một phần của nội dung giao dịch.
  - Tra cứu thông tin theo khoảng giá trị cụ thể: tra cứu các giao dịch diễn ra trong một khoảng thời gian hoặc số tiền giao dịch nằm trong một khoảng giá trị nào đó.
- Xây dựng các cấu trúc dữ liệu cho việc tìm kiếm

2. Tổng quan lý thuyết

2.1. Ý tưởng giải quyết bài toán

- Với mỗi nội dung cần tìm kiếm, duy trì một cấu trúc dữ liệu hỗ trợ riêng cho việc tìm kiếm trên nội dung đó.
- Khi đó chúng ta chỉ mất nhiều thời gian cho việc tạo cấu trúc dữ liệu thay vì mất thời gian tìm kiếm tuần tự trên nội dung đó.
- Yêu cầu tìm kiếm dữ liệu phức hợp (tìm kiếm trên nhiều nội dung), kết quả là phép intersection giữa các kết quả tìm kiếm trên từng nội dung.

2. Tổng quan lý thuyết

2.2. Các cấu trúc dữ liêu hỗ trợ giải quyết bài toán

2.2.1. Cây AVL

a. Đinh nghĩa

Cây AVL là một cây tìm kiếm nhị phân tự cân bằng (self-balancing Binary Search Tree), trong đó chiều cao của cây con trái và chiều cao của cây con phải một nút bất kì khác nhau không quá 1.

b. Cấu trúc

Cây AVL là một cây tìm kiếm nhị phân cân bằng (Balanced Binary Search Tree), nên cấu trúc của nó cũng giống cấu trúc của một cây tìm kiếm nhị phân tự cân bằng

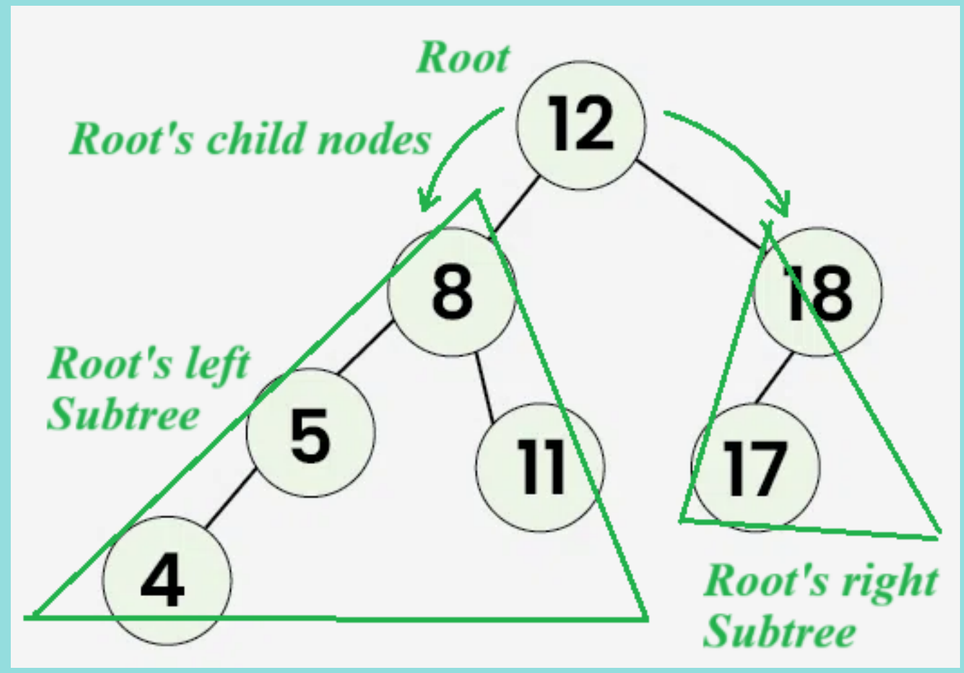
2. Tổng quan lý thuyết

2.2. Các cấu trúc dữ liêu hỗ trợ giải quyết bài toán

2.2.1. Cây AVL

b. Cấu trúc

- *Nút gốc (Root)*
- *Nút con (Child Nodes)*
- *Cây con trái (Left Subtree)*
- *Cây con phải (Right Subtree)*
- *Chiều cao của cây (height)*
- *Hệ số cân bằng:*



$$BF(N) = height(Nleft) - height(Nright)$$

2. Tổng quan lý thuyết

2.2. Các cấu trúc dữ liêu hỗ trợ giải quyết bài toán

2.2.1. Cây AVL

c. Đặc điểm

- *Cây AVL cân bằng chiều cao, sự chênh lệch giữa cây con trái và cấy con phải của bất kì nút nào không được vượt quá 1*
- *Các thao tác tìm kiếm, chèn và xóa đều có độ phức tạp là O(log(N))*

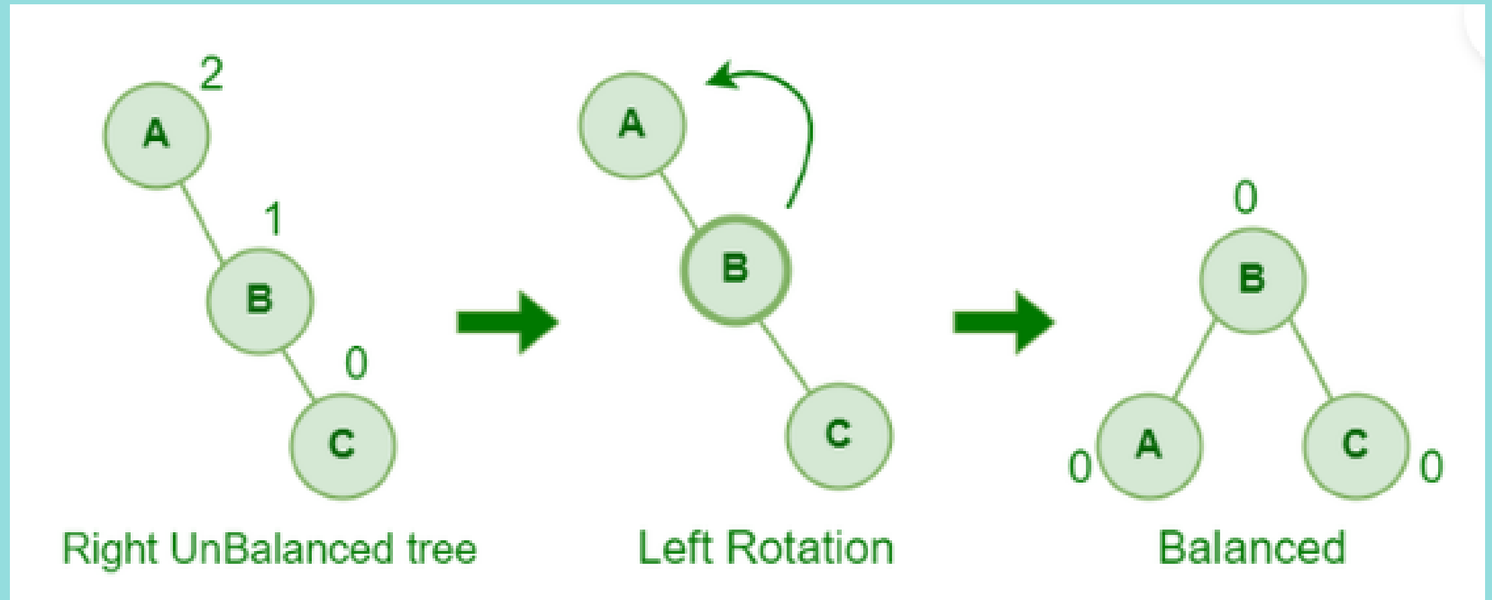
2. Tổng quan lý thuyết

2.2. Các cấu trúc dữ liêu hỗ trợ giải quyết bài toán

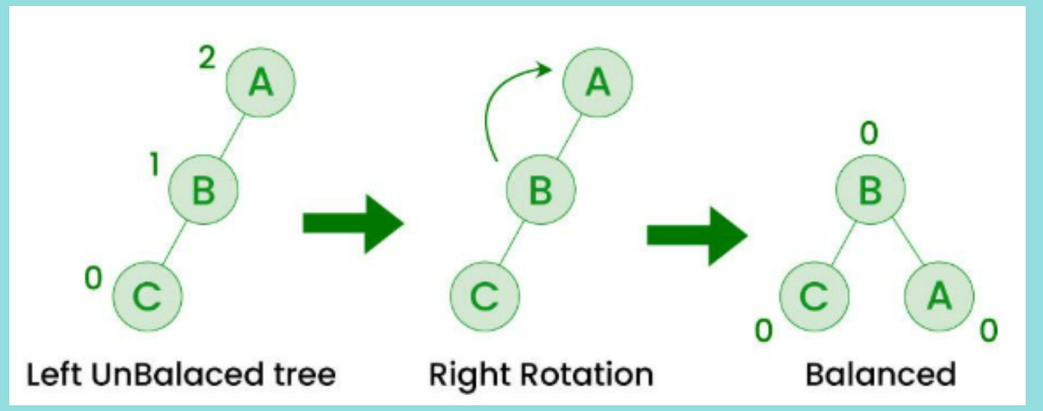
2.2.1. Cây AVL

d. Các thao tác trên cây AVL

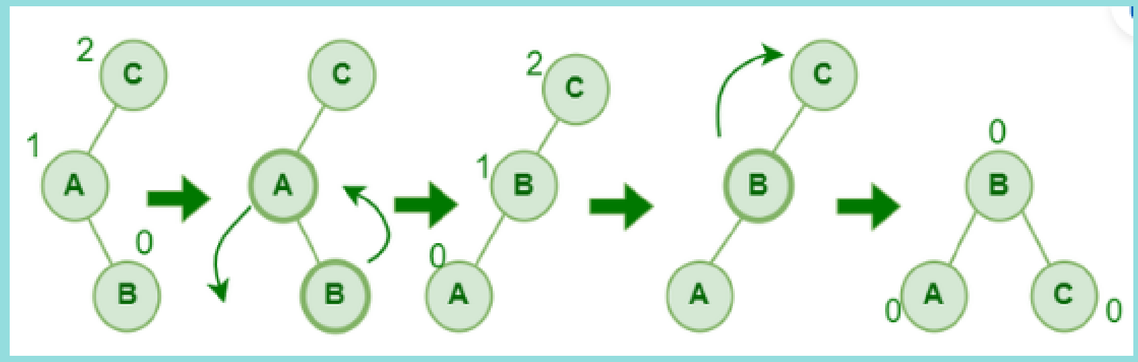
- ***Xoay các cây con***: cây AVL có thể xoay theo 4 cách
  - ***Xoay trái (Left Rotation)***:



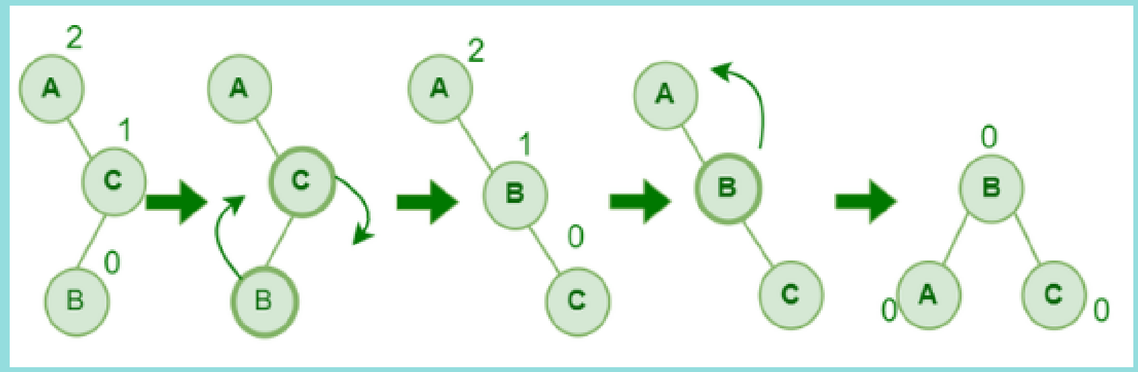
- ***Xoay các cây con***: cây AVL có thể xoay theo 4 cách
  - ***Xoay phải (Right Rotation)***:



  - ***Xoay trái-phải (Left-Right Rotation)***



  - ***Xoay phải-trái (Right-Left Rotation)***

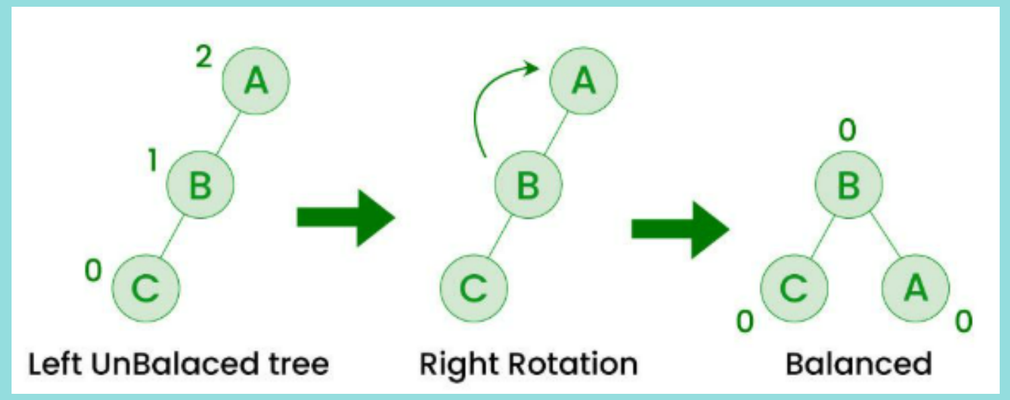


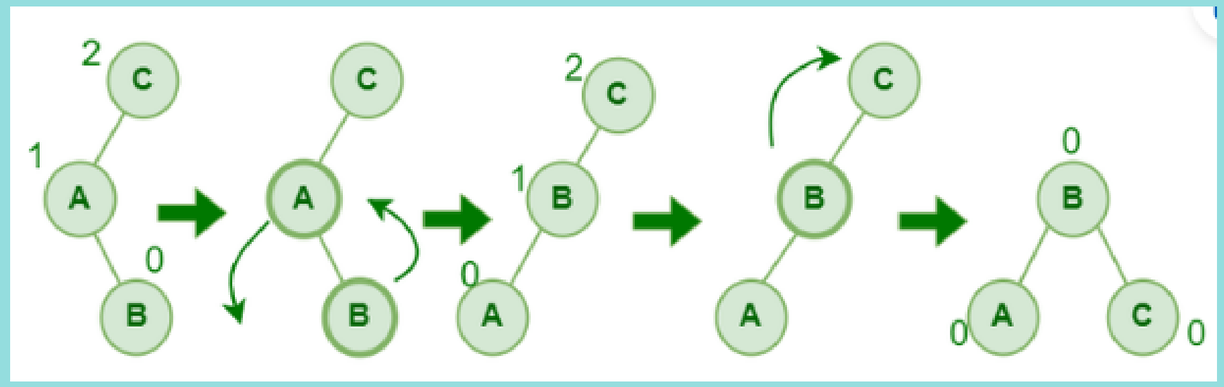
- ***Chèn***: chèn thêm nút mới vào cây AVL

1. Đặt nút mới được chèn vào là ***w***. Thực hiện chèn BST chuẩn cho ***w***.
2. Bắt đầu từ ***w***, đi lên và tìm *nút mất cân bằng đầu tiên*. Giả sử:
    - **z** là nút mất cân bằng đầu tiên,
    - **y** là nút con của **z**
    - **x** là nút cháu của **z**.
3. Cân bằng lại cây bằng cách thực hiện phép quay thích hợp trên cây con có gốc là **z**. Có thể có 4 trường hợp có thể xảy ra cần được xử lý:

- ***Chèn***: chèn thêm nút mới vào cây AVL

3. Có thể có 4 trường hợp có thể xảy ra cần được xử lý:
    - **y** là con trái của **z** và **x** là con trái của **y** (trường hợp trái trái) → Thực hiện phép xoay phải
    - **y** là con trái của **z** và **x** là con phải của **y** (trường hợp trái-phải) → Thực hiện phép xoay trái-phải

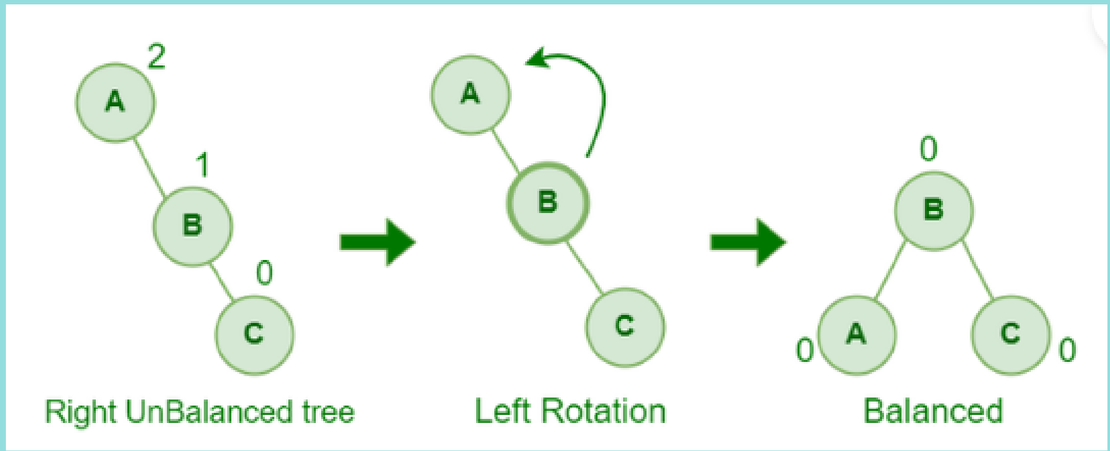


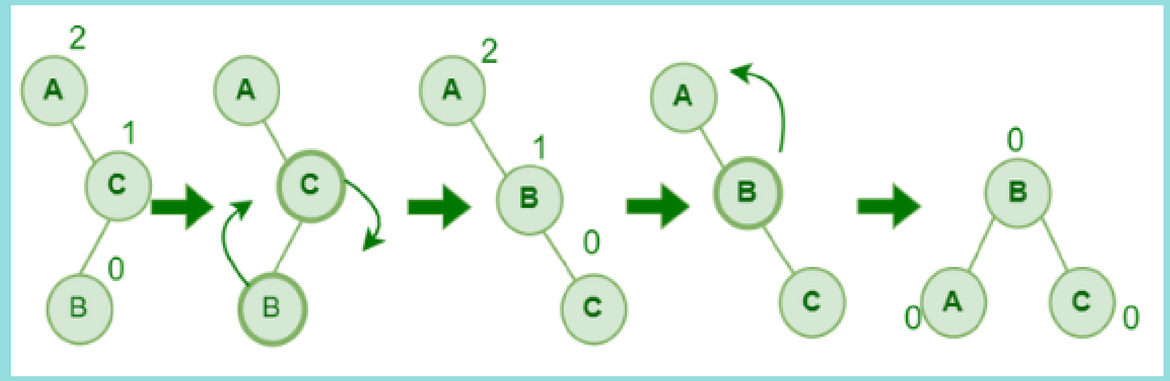


- ***Chèn***: chèn thêm nút mới vào cây AVL

3. Có thể có 4 trường hợp có thể xảy ra cần được xử lý:
    - **y** là con phải của **z** và **x** là con phải của **y** (trường hợp phải phải)

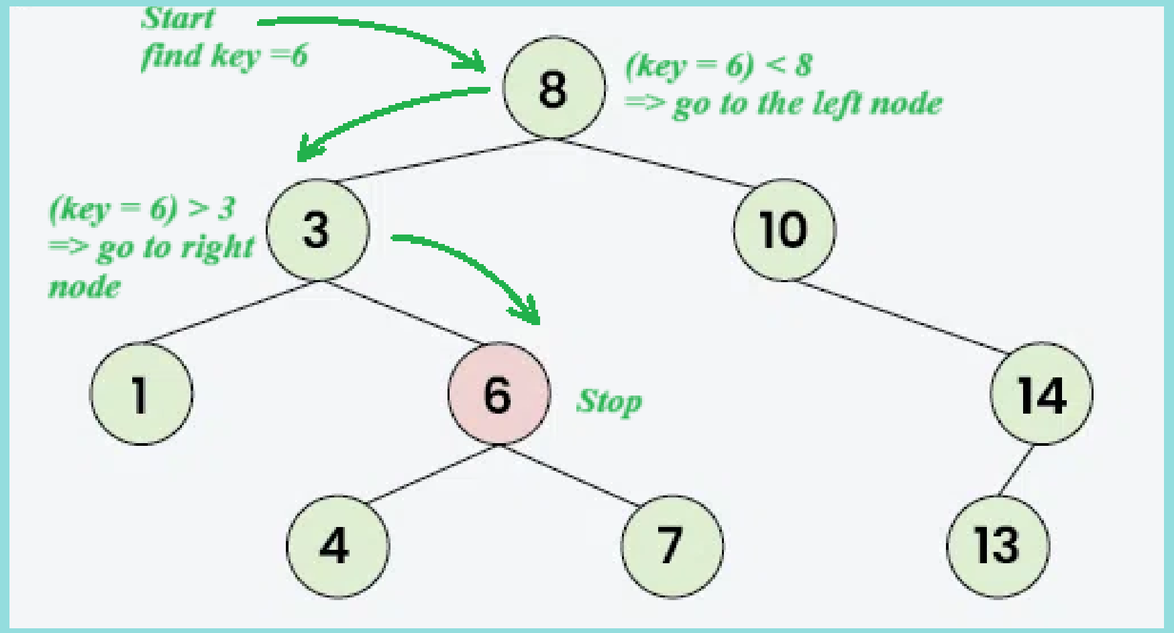
      → Thực hiện phép xoay trái
    - **y** là con phải của **z** và **x** là con trái của **y** (trường hợp phải-trái) → Thực hiện phép xoay phải-trái





- ***Tìm kiếm:*** quá trình thực hiện tìm kiếm một nút có khóa cho trước giống như trong BST. Giả sử chúng ta muốn tìm nút có khóa **X**, chúng ta bắt đầu từ gốc. Sau đó:

◦ So sánh giá trị cần tìm kiếm với khóa của nút gốc. Nếu bằng nhau thì hoàn tất việc tìm kiếm, nếu nhỏ hơn thì chuyển đến cây con bên trái; ngược lại chuyển sang cây con bên phải.
◦ Lặp lại các bước trên cho đến khi không thể duyệt cây được nữa
◦ Nếu tại bất kỳ lần lặp nào, tìm thấy khóa, trả về nút tìm kiếm được. Nếu không thì trả về null.



- ***Duyệt cây trung thứ tự trong khoảng giá trị (n1, n2):*** Đối với một nút nhất định, nếu giá trị của khóa lớn hơn ***n2***, thì đệ quy gọi vào cây con bên trái và nếu giá trị của nút nhỏ hơn ***n1***, thì xử lý cây con bên phải

2. Tổng quan lý thuyết

2.2. Các cấu trúc dữ liêu hỗ trợ giải quyết bài toán

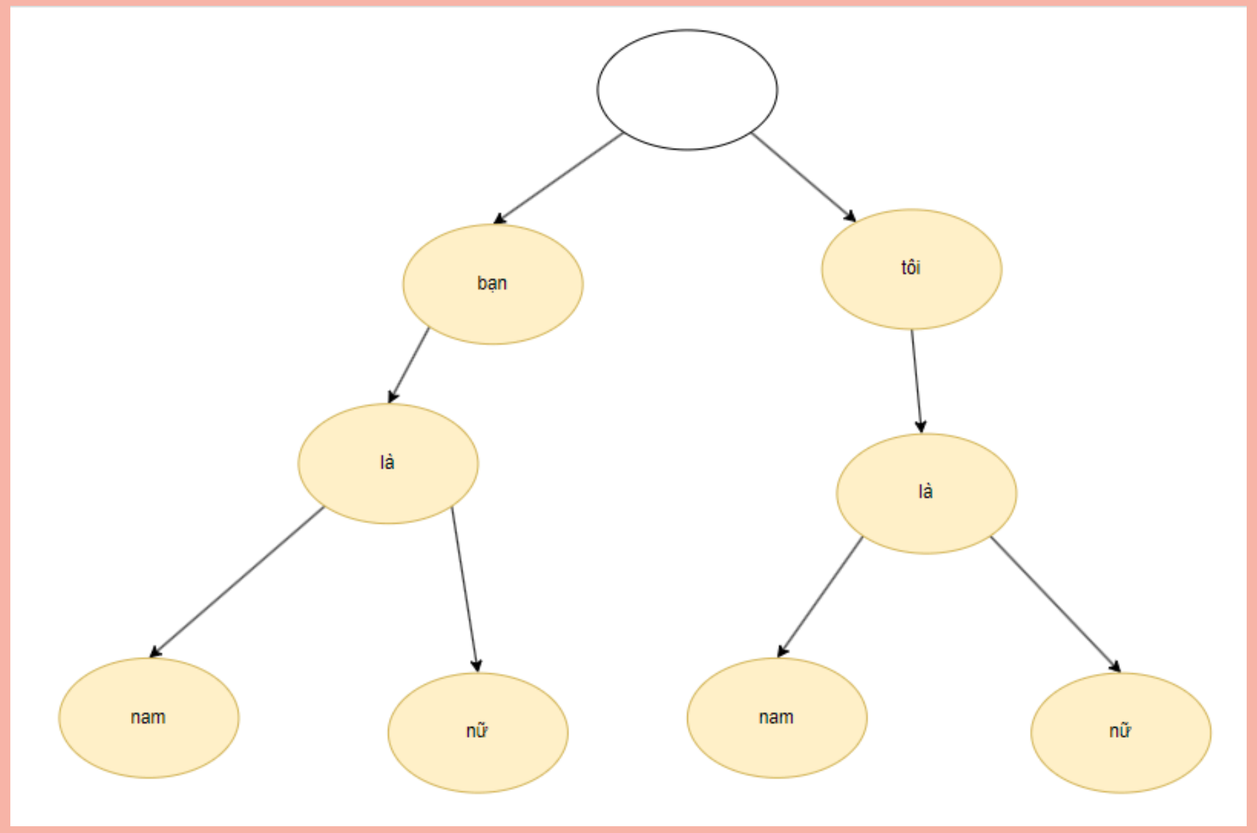
2.2.2. Cây tiền tố (Trie Tree)

a. Đinh nghĩa

Là một cấu trúc dữ liệu dạng cây được sử dụng phổ biến để quản lý và tìm kiếm các dữ liệu dạng chuỗi. Trie đặc biệt hiệu quả trong việc lưu trữ các từ hoặc câu và thực hiện các thao tác tìm kiếm tiền tố nhanh chóng.

b. Đặc điểm

- 1 cây có gốc là nút rỗng
- Mỗi nút trong Trie đại diện cho một phần trong chuỗi. (1 word)
- Đường dẫn từ gốc đến nút biểu thị tiền tố của chuỗi được lưu trữ trong Trie.
- Các chuỗi có tiền tố giống nhau sẽ có chung các nút tương ứng.
- Trie thường được sử dụng trong các ứng dụng như kiểm tra chính tả, tự động hoàn thành (autocomplete), và tìm kiếm chuỗi.



Cấu trúc Trie dùng để truy vấn theo detail

Trie Node

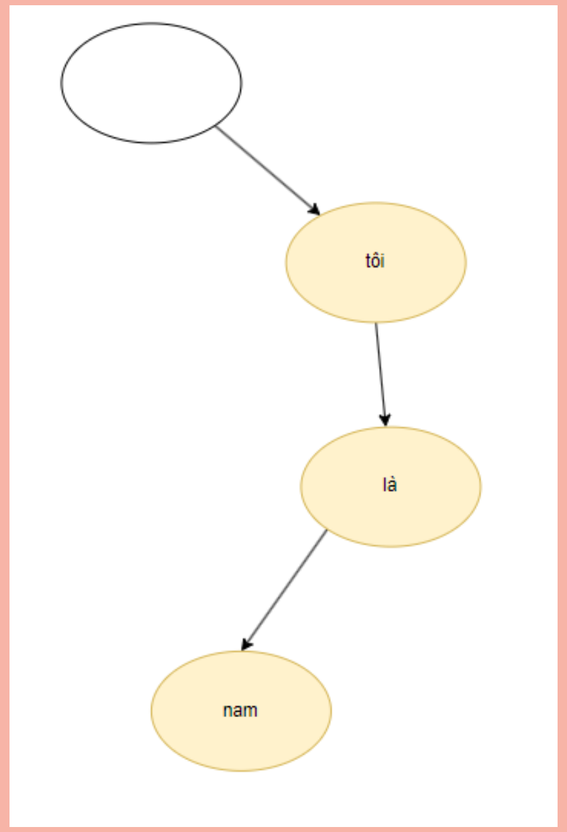
- children: Một đối tượng (object) lưu trữ các nút con, mỗi nút con được ánh xạ theo từ khóa (word)
- transactions: Một mảng lưu trữ các thông tin liên quan đến giao dịch (transaction) chứa đoạn substring được biểu diễn bởi nhánh từ gốc đến nút hiện tại.

```
class TrieNode {
    constructor() {
        this.children = {};
        this.transactions = [];
    }
}
```

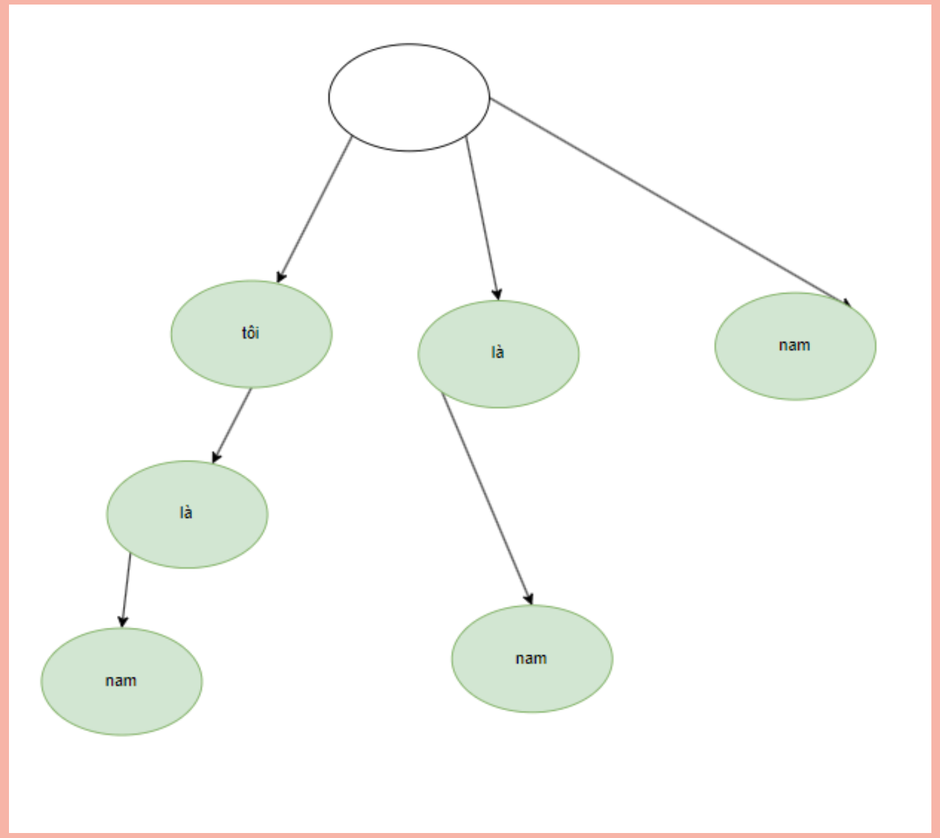
c. Chức năng insert

```
insert(sentence, transaction) {
    sentence=sentence.toLowerCase();
    // Chia câu thành các từ con
    const words = sentence.split(/[\s\-,]+/);
    // Với mỗi từ, chèn toàn bộ các đoạn substring bắt đầu bằng từ đó vào cây.
    for (let i = 0; i < words.length; i++) {
        let node = this.root;
        for (let j = i; j < words.length; j++) {
            // Với mỗi đoạn substring, bắt đầu từ nút gốc,
            // nếu đoạn substring không có trong cây, thêm nút mới.
            const word = words[j];
            if (!node.children[word]) {
                node.children[word] = new TrieNode();
            }
            node = node.children[word];
            // Đối với mỗi đoạn substring trong câu, lưu trữ toàn bộ
            // thông tin giao dịch (transaction) tương ứng tại các nút trong cây.
            node.transactions.push(transaction);
        }
    }
}
```

c. So sánh với Trie thông thường



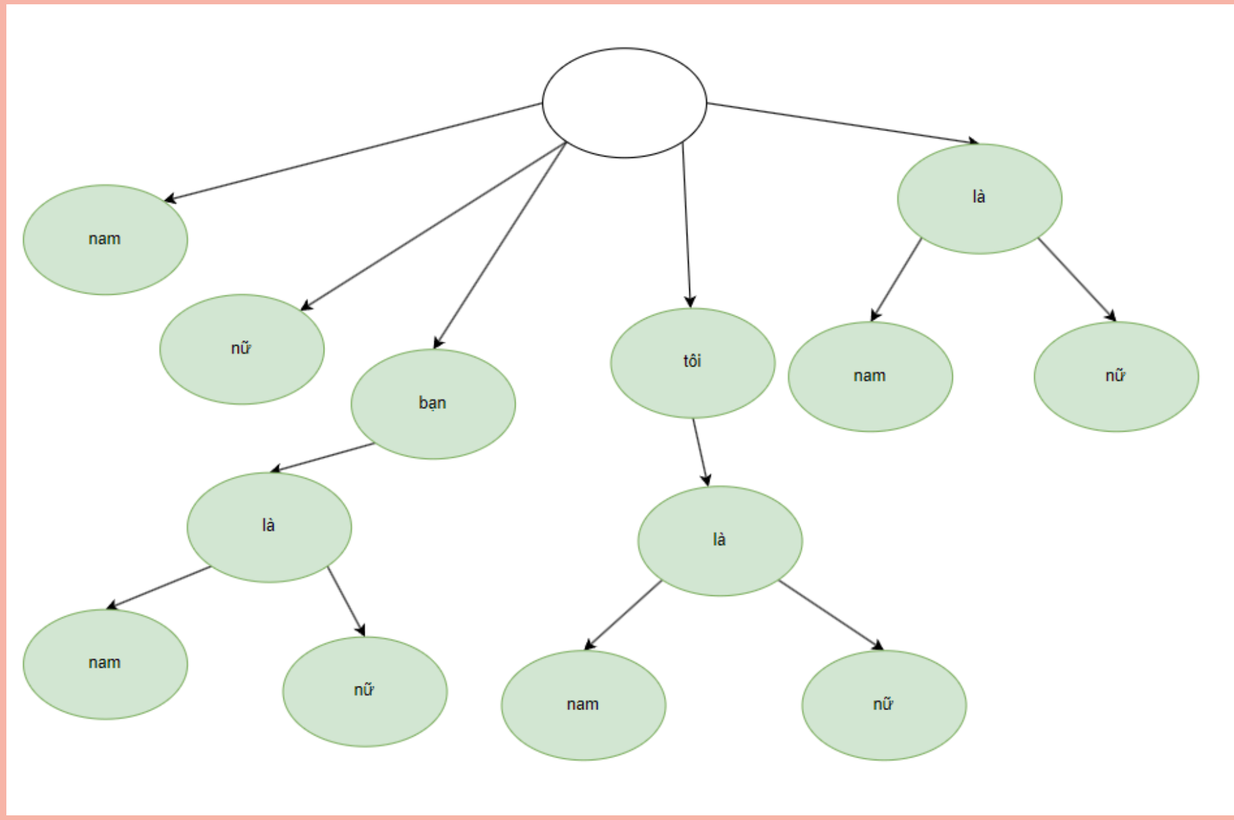
Độ phức tạp trung bình: ***O(n.k)*** với n là số câu, k là số từ trung bình mỗi câu
Chỉ cho phép tìm kiếm theo tiền tố



Độ phức tạp trung bình: ***O(n.k^2)*** với n là số câu, k là số từ trung bình mỗi câu
Cho phép tìm kiếm với mọi đoạn substring

c. Chức năng search

```
search(substring) {
    // làm chuẩn lại chuỗi con
    substring=substring.toLowerCase();
    const words = substring.split(/[\s\-,]+/);
    let node = this.root;
    // Duyệt qua các nút trong Trie theo từng từ con.
    // Nếu một từ không tồn tại trong Trie, trả về mảng rỗng ([]).
    for (const word of words) {
        if (!node.children[word]) {
            return [];
        }
        node = node.children[word];
    }
    // Nếu tìm thấy, trả về tất cả các giao dịch
    // (transactions) liên quan tại nút cuối cùng.
    return node.transactions;
}
```



Độ phức tạp khi tìm kiếm: O(k) với k là độ dài trung bình chuỗi (không đổi so với Trie thông thường) --> phù hợp với yêu cầu

3. Thiết kế hệ thống

3.1. Ứng dụng cấu trúc dữ liệu hỗ trợ tìm kiếm

- Các thao tác tìm kiếm giao dịch:
  - Diễn ra trong ngày cụ thể; trong khoảng thời gian.
  - Với số tiền cụ thể; số tiền năm trong khoảng nhất định.
  - Có nội dung/một phần nội dung.
  - Điều kiện là sự kết hợp của các điều kiện trên

- Để hỗ trợ quá trình tìm kiếm, chúng ta
  - Duy trì 3 cây AVL cho các cột: ngày giờ (date-time), số tiền gửi (credit) và số tiền nợ (debit) với các khóa của nút lần lượt là ngày giờ, số tiền gửi và số tiền nợ;
  - Duy trì cây tiền tố cho cột nội dung.

3. Thiết kế hệ thống

3.1. Ứng dụng cấu trúc dữ liệu hỗ trợ tìm kiếm

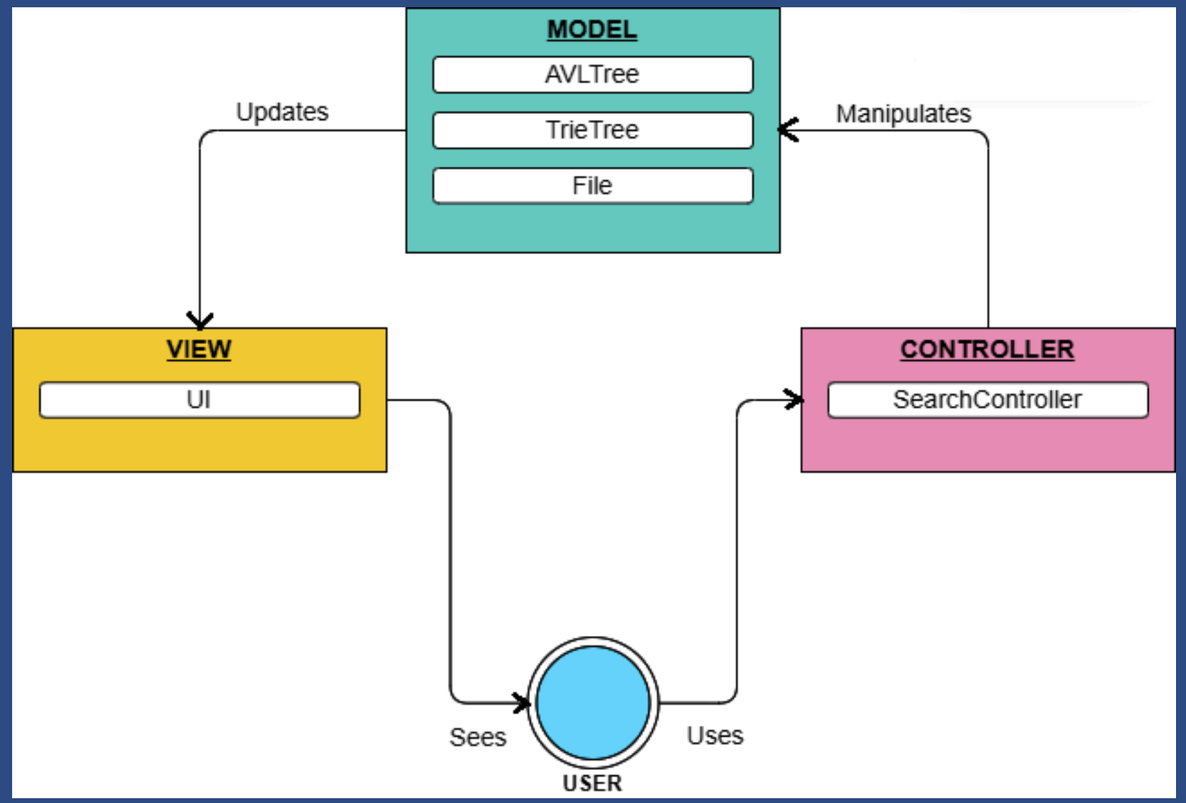
- Qúa trình xử lí dữ liệu như sau:
  - ***Chuẩn bị dữ liệu***: đọc từng bảo ghi giao dịch trong dữ liệu sao kê, với mỗi giao dịch thực hiện chèn (insert) cặp dữ liệu (khóa; thứ tự bản ghi) vào cấu trúc dữ liệu phù hợp
  - ***Tìm kiếm dữ liệu***: thực hiện tìm kiếm (search) trên cấu trúc dữ liệu phù hợp; kết quả thu được thực hiện gioa với nhua nếu truy vấn kết hợp nhiều điều kiện trên các cột nội dung khác nhau.

3. Thiết kế hệ thống

3.2. Kiến trúc hệ thống

1. ***Models***:

- Chịu trách nhiệm quản lý/ tương tác với dữ liệu của ứng dụng
- Chứa các quy tắc nghiệp vụ và logic điều kiển cách dữ liệu dược tạo, lưu trữ và sửa đổi
- Chứa các module:
  - ***AVLTree***, ***TrieTree***: chứa các cấu trúc dữ liệu hỗ trợ việc tìm kiếm dữ liệu
  - ***File***: quản lí/tương tác với file dữ liệu sao kê

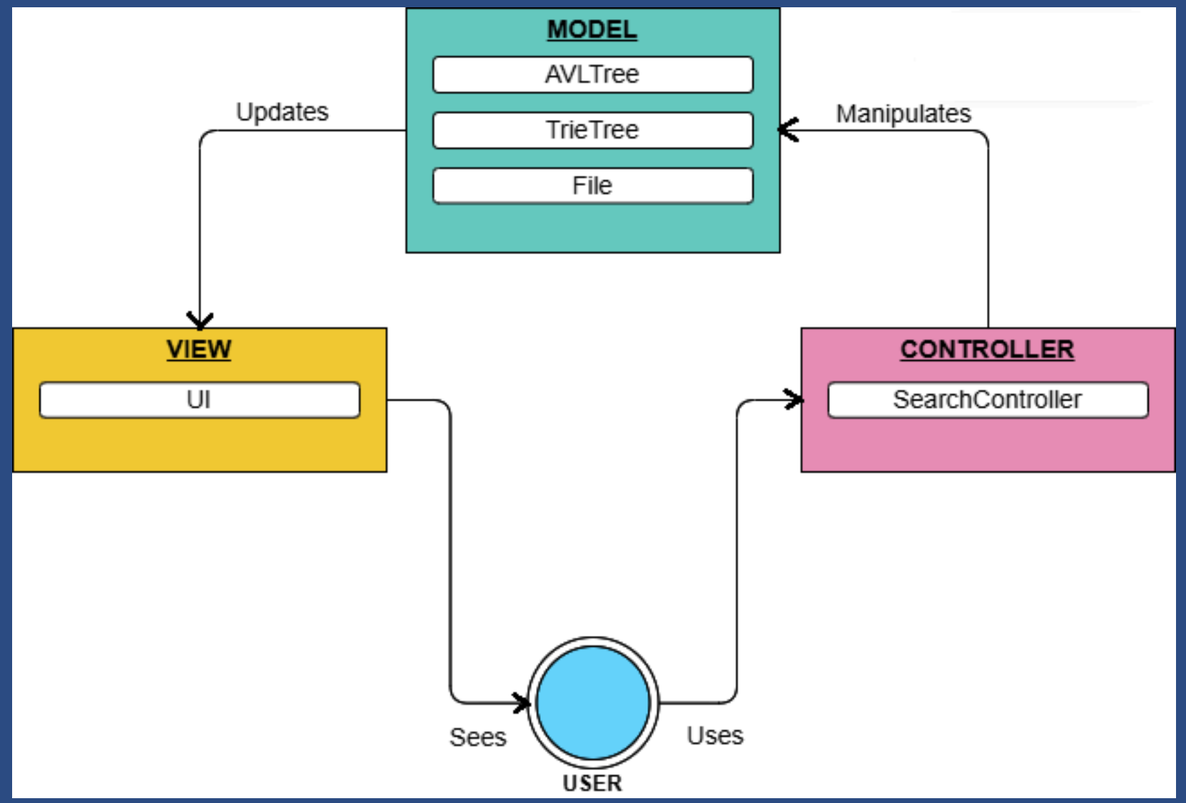


3. Thiết kế hệ thống

3.2. Kiến trúc hệ thống

2. *Views*:

- Chịu trách nhiệm hiển thị dữ liệu cho người dùng.
- Lấy dữ liệu từ Model và định dạng nó để hiển thị cho người dùng.
- Tương tác người dùng: View xử lý các yếu tố giao diện người dùng như văn bản, hình ảnh và nút bấm, và cập nhật hiển thị dựa trên các thay đổi trong Mode

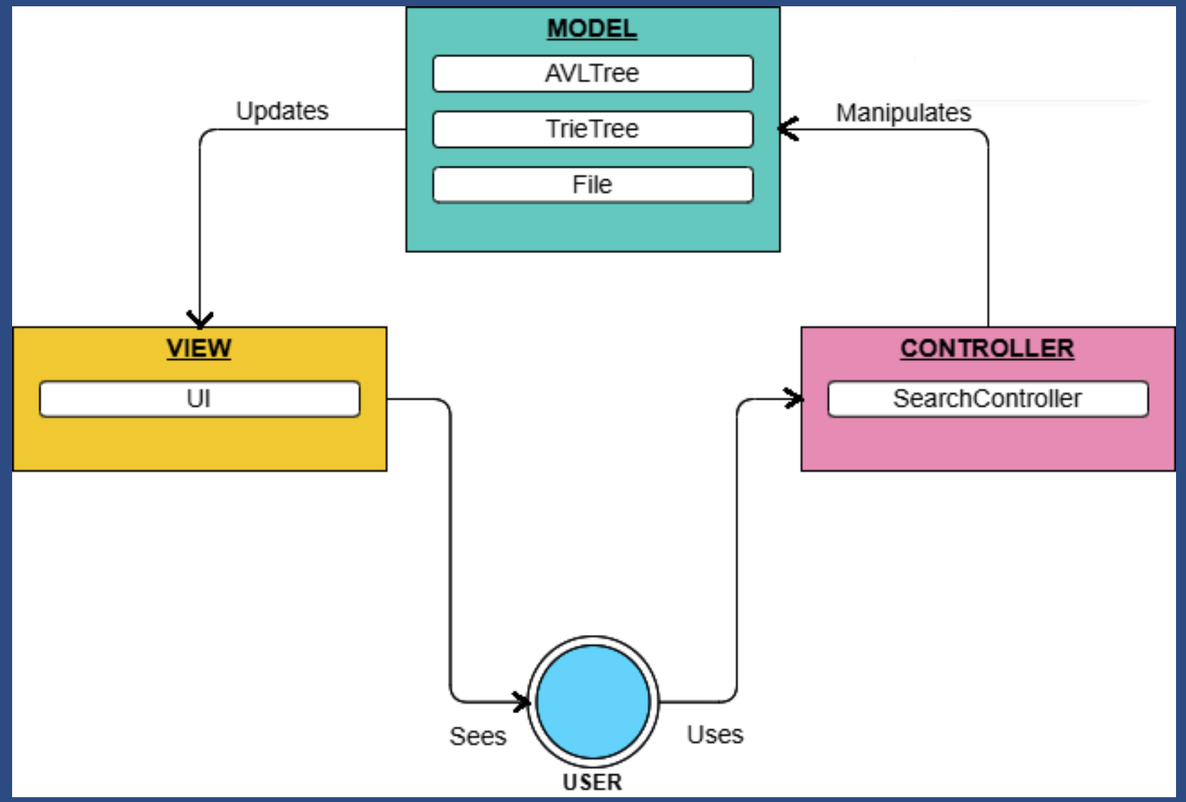


3. Thiết kế hệ thống

3.2. Kiến trúc hệ thống

3. Controllers:

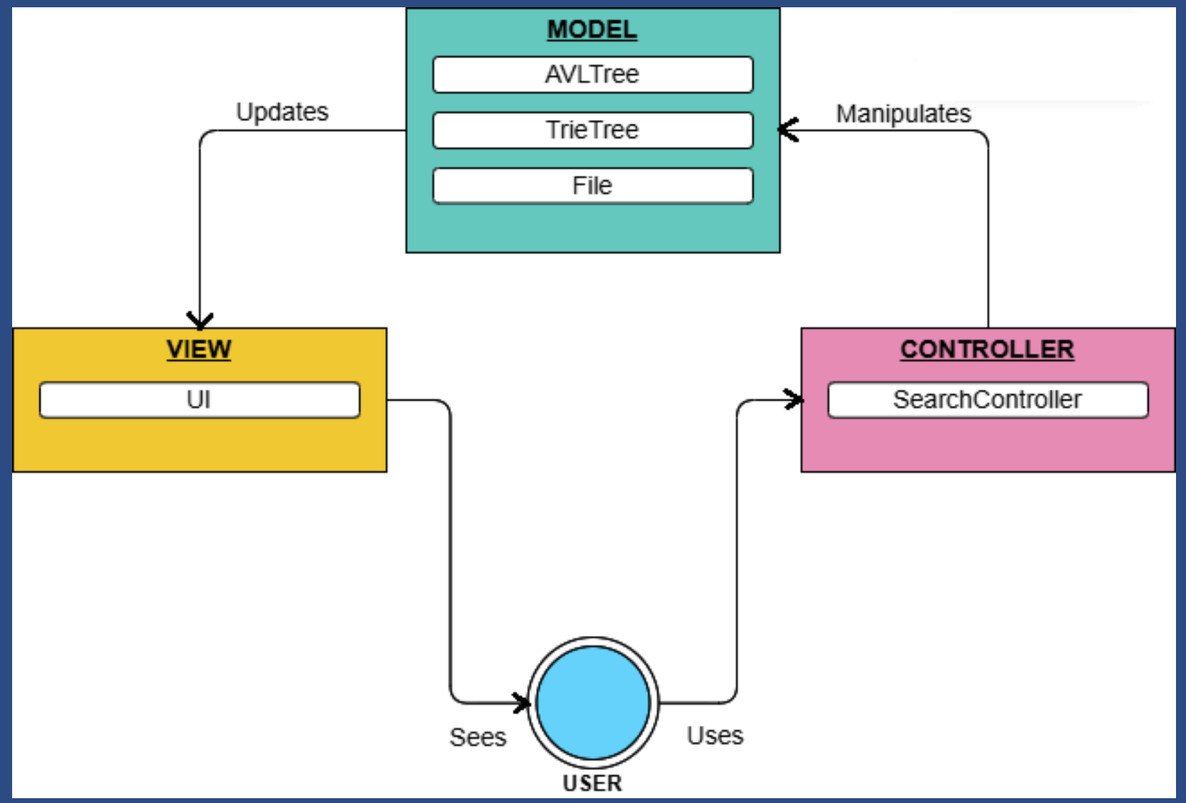
- Chịu trách nhiệm xử lý đầu vào và tương tác của người dùng. Nó hoạt động như một trung gian giữa Model và View.
- Xử lý các đầu vào của người dùng, như nhấp chuột và gửi biểu mẫu, và cập nhật Model tương ứng.



3. Thiết kế hệ thống

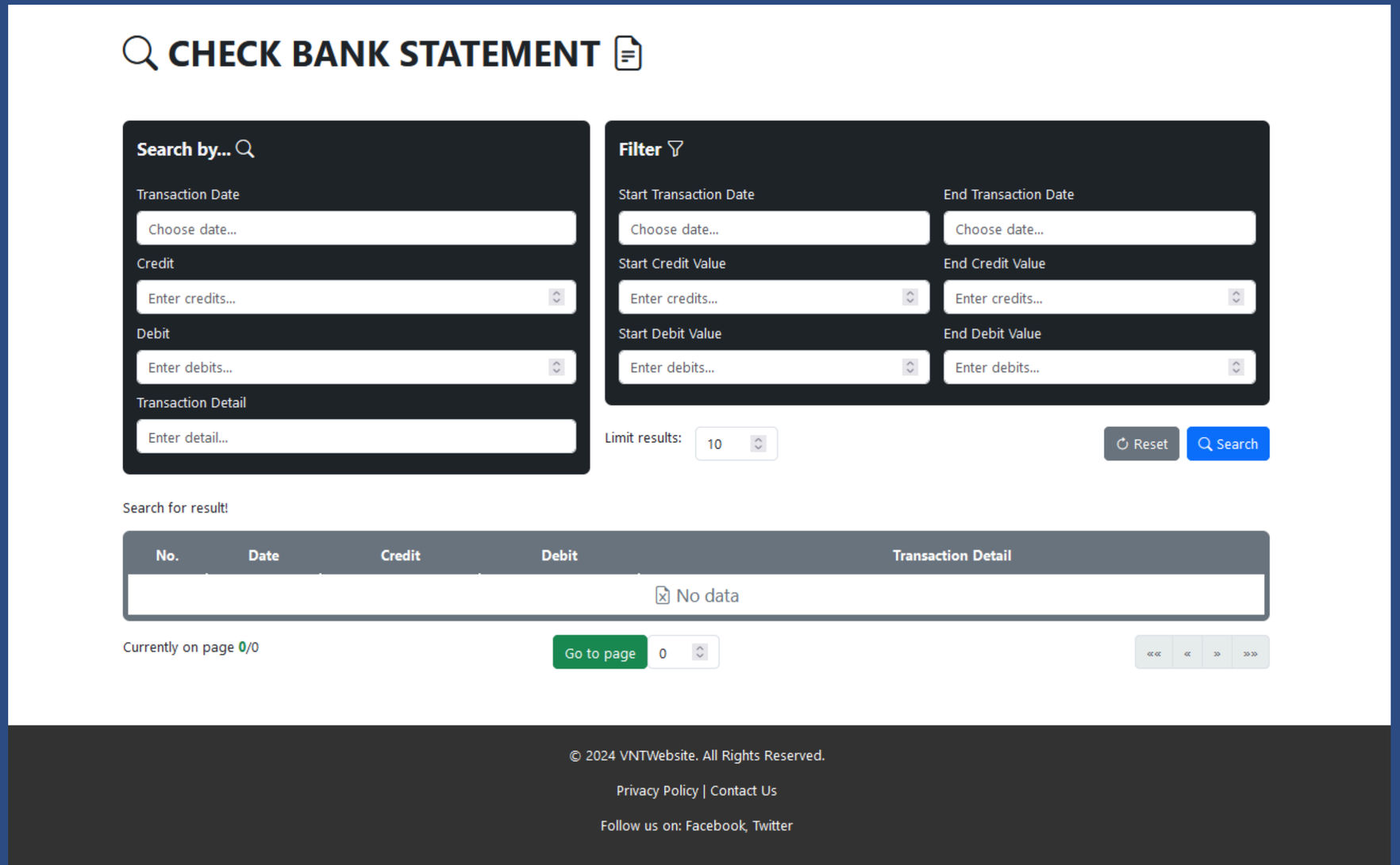
3.2. Kiến trúc hệ thống

- ***Tương tác người dùng***: Người dùng tương tác với View, View gửi đầu vào đến Controller.
- ***Xử lý controller***: Controller xử lý đầu vào và cập nhật Model.
- ***Cập nhật Model***: Model cập nhật trạng thái dựa trên hướng dẫn của Controller.
- ***Làm mới View***: View lấy dữ liệu cập nhật từ Model và làm mới hiển thị để phản ánh các thay đổi.



3. Thiết kế hệ thống

3.3. Giao diện người dùng



3. Thiết kế hệ thống

3.4. API Endpoints

1. API tìm kiếm dữ liệu:

- *Mô tả*: API tìm kiếm tất cả các giao dịch thảo điều kiện tìm kiếm
- *URL*: /search
- *Phương thức HTTP*: GET
- *Tham số*: Tham số truy vấn:
  - dateSearch, fromDate, toDate
  - creditSearch, fromCredit, toCredit
  - debitSearch, fromDebit, toDebit
  - detailSearch
  - limit
- Cấu trúc phản hồi:
  - ids: mảng các Id giao dịch thỏa mãn điều kiện tìm kiếm
  - data: mảng chứa nội dung giao dịch chi tiết. Độ dài của mảng không vượt quá limit

3. Thiết kế hệ thống

3.4. API Endpoints

2. API truy xuất dữ liệu:

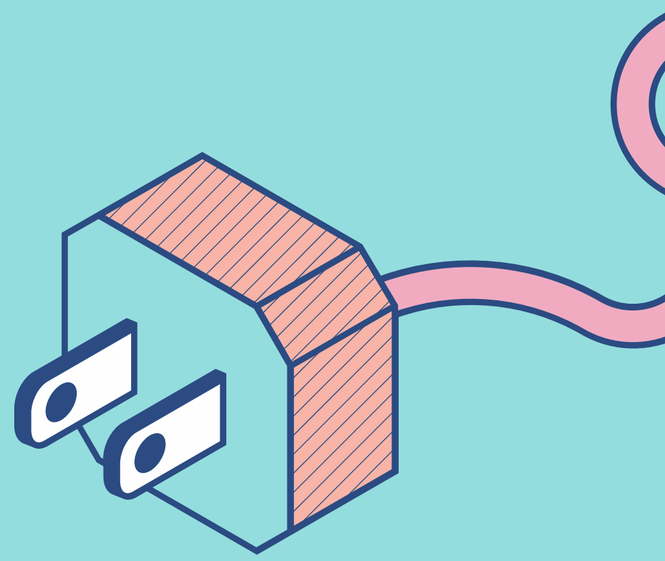
- *Mô tả*: API truy xuất dữ liệu giao dich dựa trên Id của giao dịch
- *URL*: /search/get
- *Phương thức HTTP*: GET
- *Tham số*: Tham số truy vấn:
  - ids: chuỗi các Id của giao dịch, cách nhau bởi dấu ","
- Cấu trúc phản hồi:
  - data: mảng chứa nội dung giao dịch chi tiết.

4. Hiện thực

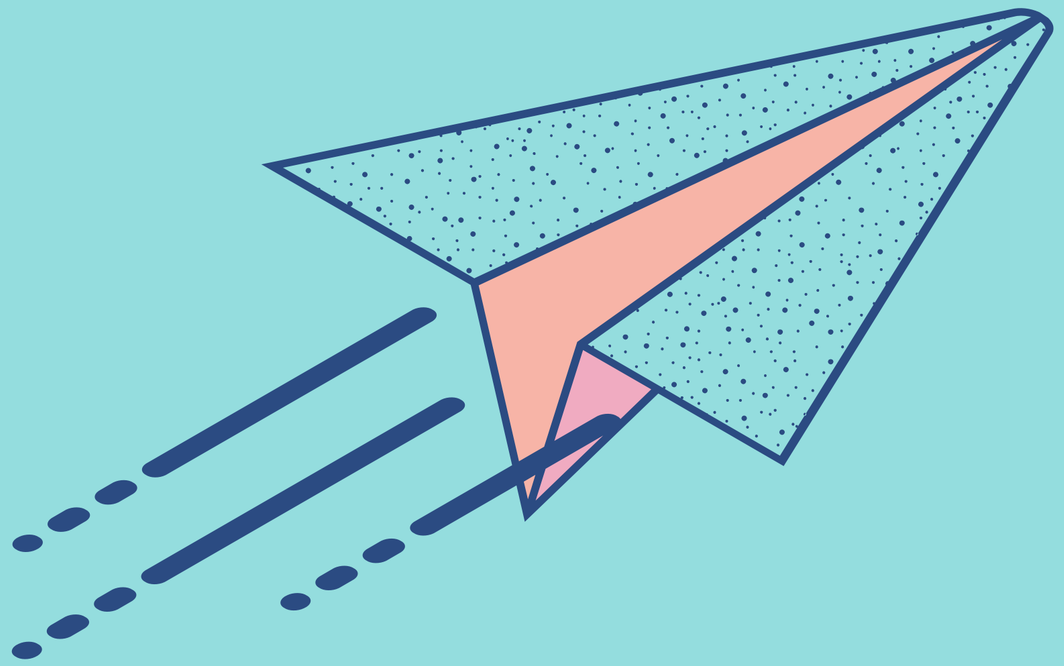
- ***Express*** là một framework web ứng dụng cho Node.js, được thiết kế để xây dựng các ứng dụng web và API một cách nhanh chóng và dễ dàng

- ***Handlebars (HBS)*** là một engine template cho phép bạn tạo giao diện người dùng động trong các ứng dụng web

- ***Bootstrap:*** một framework front-end mã nguồn mở, được sử dụng để phát triển các giao diện web hiện đại và đáp ứng (responsive). Dưới đây là một tổng quan về Bootstrap:



5. Demo sản phẩm



Thanks for listening