BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

ĐỒ ÁN MÔN HỌC

CÂY TIỀN TỐ
(TRIE TREE)

Giảng viên hướng dẫn	:	ThS. Nguyễn Thanh Sơn
Sinh viên thực hiện 1	:	Hứa Mạnh Tân
Mã sinh viên 1	:	23521396
Sinh viên thực hiện 2	:	Nguyễn Tấn Tài
Mã sinh viên 2	:	23521376
Lớp	:	CS523.P21
Bộ môn	:	CTDL&GTNC

TP. HỒ CHÍ MINH, THÁNG 3 NĂM 2025

MỤC LỤC

NHIỆM VỤ ĐỒ ÁN MÔN HỌC

1. Nghiên cứu và phân tích cấu trúc dữ liệu cây tiền tố (trie tree)

   + Tìm hiểu khái niệm **cây tiên tố** và các thuật toán liên quan

   + Phân tích cách sinh và lưu trữ các số nguyên tố hiệu quả

2. Thiết kế cấu trúc dữ liệu

   + Xây dựng cấu trúc dữ liệu để biểu diễn cây tiên tố.

   + Xác định cách tổ chức các nút và mối quan hệ giữa chúng.

3. Cài đặt thuật toán

   + Viết thuật toán để tạo cây từ danh sách các số nguyên tố.

   + Hiện thực các thao tác cơ bản trên cây (thêm, xóa, tìm kiếm).

4. Thiết kế website từ điển ứng dụng

   + Thiết kế giao diện web phù hợp

   + kết hợp cơ sở dữ liệu để hoàn chỉnh website

LỜI CẢM ƠN

Trước hết, em xin gửi lời cảm ơn chân thành đến trường Đại học Công nghệ Thông tin – ĐHQG TP.HCM, nơi đã tạo điều kiện cho em được học tập và nghiên cứu trong một môi trường đầy đủ cơ sở vật chất và kiến thức chuyên môn.

Em xin bày tỏ lòng biết ơn sâu sắc đến Ths Nguyễn Thanh Sơn , người đã tận tình hướng dẫn, truyền đạt kiến thức, góp ý và động viên em trong suốt quá trình thực hiện đồ án. Sự tận tâm và những lời chỉ dạy quý báu của thầy/cô đã giúp em hoàn thiện bài làm của mình một cách tốt nhất.

Bên cạnh đó, em cũng xin gửi lời cảm ơn đến các thầy (cô) khoa khoa học máy tính đã cung cấp cho em những kiến thức nền tảng quan trọng, làm hành trang vững chắc để em có thể thực hiện đồ án này.

Mặc dù đã cố gắng hoàn thiện đồ án một cách tốt nhất, nhưng không thể tránh khỏi những thiếu sót . Em rất mong nhận được những góp ý quý báu từ thầy cô và các bạn để có thể hoàn thiện hơn trong tương lai.

Em xin chân thành cảm ơn!

Hứa Mạnh Tân
Nguyễn Tấn Tài
TPHCM,ngày 20 tháng 3 năm 2025

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Tp.HCM, ngày … tháng … năm …

GVHD

Phần 1 : Giới thiệu đề tài

1.1 Tên đề tài

- Tiếng Anh : Trie tree
- Tiếng việt : Cây tiền tố

1.2 Mô tả đề tài

- Trie tree (cây tiền tố) là một cấu trúc dữ liệu dạng cây được sử dụng rộng rãi trong các ứng dụng tìm kiếm chuỗi, đặc biệt là trong từ điển và hệ thống gợi ý từ . Đề tài này tập trung vào việc xây dựng một trang web từ điển sử dụng trie tree với ngôn ngữ chính là node.js giúp tra cứu từ nhanh chóng , gợi ý từ theo tiền tố và hỗ trợ tiềm kiếm với độ chính xác cao
- Bên cạnh xây dựng trang web đề tài cũng sẽ mô phỏng thuật toán trie tree một cách cụ thể rõ ràng và dễ dàng tiếp cận với sinh viên , lập trình viên ,…với hình ảnh sinh động cụ thể ở phần trình bày giúp người học có cái nhìn trực quan hơn về trie tree từ đó có thể áp dụng vào các bài toán cụ thể như gợi ý từ , tìm kiếm từ , kiểm tra từ ,…

1.3 Các chức năng của đề tài

- Thêm từ vào TrieTree
- Tìm kiếm từ trong TrieTree
- Xóa từ khỏi TrieTree
- Gợi ý từ theo tiền tố
- Giải thích thuật toán bằng hình ảnh và đoạn code

1.4 Ngôn ngữ lập trình và ứng dụng

- Node.js (visual studio code)
- Html / Css / JavaScript (visual studio code)
- SQL (SQL server)

1.5 Các công cụ hỗ trợ khác

- Google ,Github

Phần 2 : Giới thiệu các ngôn ngữ và chức năng

2.1 Node.js

- Lịch sử : Node.js được tạo ra bởi Ryan Dahl vào năm 2009, với mục tiêu cải thiện hiệu suất của các ứng dụng web, đặc biệt là xử lý nhiều kết nối đồng thời mà không bị chặn (blocking).

- Chức năng

  - Xử lý truy vấn tìm kiếm nhanh chóng
  - Xây dựng API cho ứng dụng từ điển
  - Triển khai mô phỏng thuật toán TrieTree
  - Hỗ trợ lưu trữ và quản lý dữ liệu

2.2 Html / Css / Javascript

- Lịch sử : HTML ra đời năm 1989 bởi Tim Berners-Lee, phát triển từ HTML 1.0 (1991) đến HTML5 (2014), hỗ trợ đa phương tiện và tương tác tốt hơn. CSS xuất hiện năm 1996 để tách biệt nội dung và giao diện, với các phiên bản như CSS1, CSS2 và CSS3 giúp thiết kế web linh hoạt. JavaScript được Brendan Eich tạo ra năm 1995, ban đầu để xử lý tương tác trên trình duyệt, sau phát triển mạnh với ES6+ và Node.js. Cả ba công nghệ này kết hợp giúp xây dựng web hiện đại, từ giao diện đến logic và trải nghiệm người dùng. Ngày nay, chúng vẫn tiếp tục được cải tiến, đáp ứng nhu cầu phát triển web đa nền tảng.

- Chức năng

  - Xây dựng khung trang web
  - Định dạng giao diện
  - Tạo tương tác động
  - Giúp trang web trở nên tương tác và linh hoạt hơn

2.3 SQL

- Lịch sử : SQL ra đời từ những năm 1970, dựa trên mô hình cơ sở dữ liệu quan hệ của Edgar F. Codd và được IBM phát triển thành SEQUEL, sau này trở thành SQL. Hiện nay, SQL vẫn là tiêu chuẩn chính trong quản lý dữ liệu, với nhiều hệ quản trị như MySQL, PostgreSQL, SQL Server, và tiếp tục phát triển để hỗ trợ Big Data và AI.

- Chức năng

    - Lưu trữ và quản lý dữ liệu
    - Truy vấn và xử lý dữ liệu
    - Xác thực và quản lý người dùng
    - Tương tác với ứng dụng web
    - Tối ưu hiệu suất và bảo mật

2.4 Visual studio code

- Lịch sử : Visual Studio Code được Microsoft ra mắt năm 2015, là trình soạn thảo mã nguồn miễn phí, nhẹ, và đa nền tảng. Hiện nay, VS Code là một trong những IDE phổ biến nhất, hỗ trợ mở rộng mạnh mẽ với Extensions và tích hợp Git, Debugging, AI (Copilot)

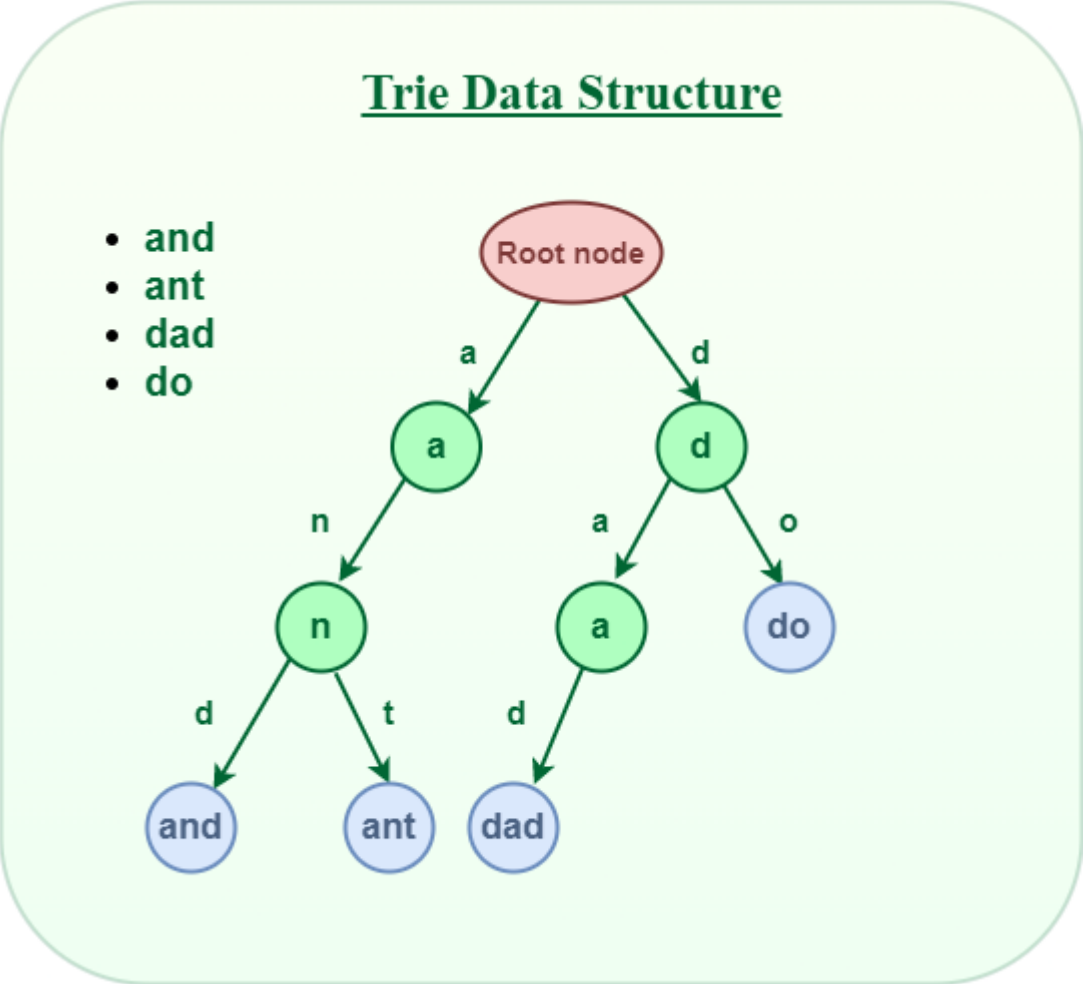
- Chức năng

    - Hỗ trợ đa ngôn ngữ
    - Tính năng Debugging
    - Hệ thống Extensions mạnh mẽ

Phần 3 : lý thuyết và mô phỏng trietree

3.1 Khái niệm

- Cấu trúc dữ liệu Trielà một cấu trúc dữ liệu dạng cây được sử dụng để lưu trữ một tập hợp chuỗi động. Nó thường được sử dụng để truy xuất và lưu trữ hiệu quả các khóa trong một tập dữ liệu lớn. Triehỗ trợ các hoạt động như chèn , tìm kiếm và xóa khóa, khiến nó trở thành một công cụ có giá trị trong các lĩnh vực như khoa học máy tính và truy xuất thông tin.



3.2 Đặc điểm

- Cấu trúc cây tiền tố – Trietổ chức dữ liệu theo dạng cây, trong đó mỗi nút đại diện cho một ký tự của từ.

- Tìm kiếm nhanh – Cho phép tra cứu từ khóa với độ phức tạp trung bình O(m) (với *m* là độ dài từ cần tìm).

- Tiết kiệm không gian (so với Hash Table) – Các tiền tố chung được dùng chung, giúp giảm bộ nhớ khi lưu trữ nhiều từ có chung ký tự đầu.

- Dễ dàng chèn, xóa từ – Các thao tác thêm và xóa có độ phức tạp O(m), nhanh hơn so với các cấu trúc dữ liệu khác trong một số trường hợp.

3.3 Cấu trúc và thuật toán

- **Mô tả nút trie**
  - Cấu trúc dữ liệu Triebao gồm các nút được kết nối bằng các cạnh.
  - Mỗi nút biểu diễn một ký tự hoặc một phần của chuỗi.
  - Nút gốc đóng vai trò là điểm bắt đầu và không lưu trữ bất kỳ ký tự nào.

```
class TrieNode {
public:

    // pointer array for child nodes of each node
    TrieNode* children[26];

    // Used for indicating ending of string
    bool isLeaf;

    TrieNode() {

        // initialize the wordEnd variable with false
        // initialize every index of childNode array with NULL
        isLeaf = false;
        for (int i = 0; i < 26; i++) {
            children[i] = nullptr;
        }
    }
};
```

- **Chèn trong Cấu trúc dữ liệu Trie**

  - Chèn “và” vào cấu trúc dữ liệu Trie:

- Bắt đầu từ nút gốc: Nút gốc không có ký tự nào được liên kết với nó và giá trị wordEnd của nó là 0 , cho biết không có từ hoàn chỉnh nào kết thúc tại điểm này.

- Ký tự đầu tiên “a”: Tính toán chỉ số bằng cách sử dụng ' a' – 'a' = 0. Kiểm tra xem child[0] có phải là null không . Vì là null, hãy tạo một TrieNode mới với ký tự “ a “, wordEnd được đặt thành 0 và một mảng con trỏ trống. Di chuyển đến nút mới này.

- Ký tự thứ hai “n”: Tính toán chỉ số bằng cách sử dụng 'n' – 'a' = 13. Kiểm tra xem child[13] có phải là null không . Có, do đó hãy tạo một TrieNode mới với ký tự “ n ”, wordEnd được đặt thành 0 và một mảng con trỏ trống. Di chuyển đến nút mới này.

- Ký tự thứ ba “d”: Tính toán chỉ số bằng cách sử dụng ' d' – 'a' = 3. Kiểm tra xem child[3] có phải là null không . Có, vì vậy hãy tạo một TrieNode mới với ký tự “ d “, wordEnd được đặt thành 1 (chỉ ra từ “ và ” kết thúc tại đây).

  - Chèn “ant” vào cấu trúc dữ liệu Trie:

- Bắt đầu từ nút gốc: Nút gốc không chứa bất kỳ dữ liệu nào nhưng nó theo dõi mọi ký tự đầu tiên của mọi chuỗi đã được chèn vào.

- Ký tự đầu tiên “a”: Tính toán chỉ số bằng cách sử dụng ' a' – 'a' = 0. Kiểm tra xem child[0] có phải là null không . Chúng ta đã có nút “ a ” được tạo từ lần chèn trước. vì vậy hãy di chuyển đến nút “ a ” hiện có.

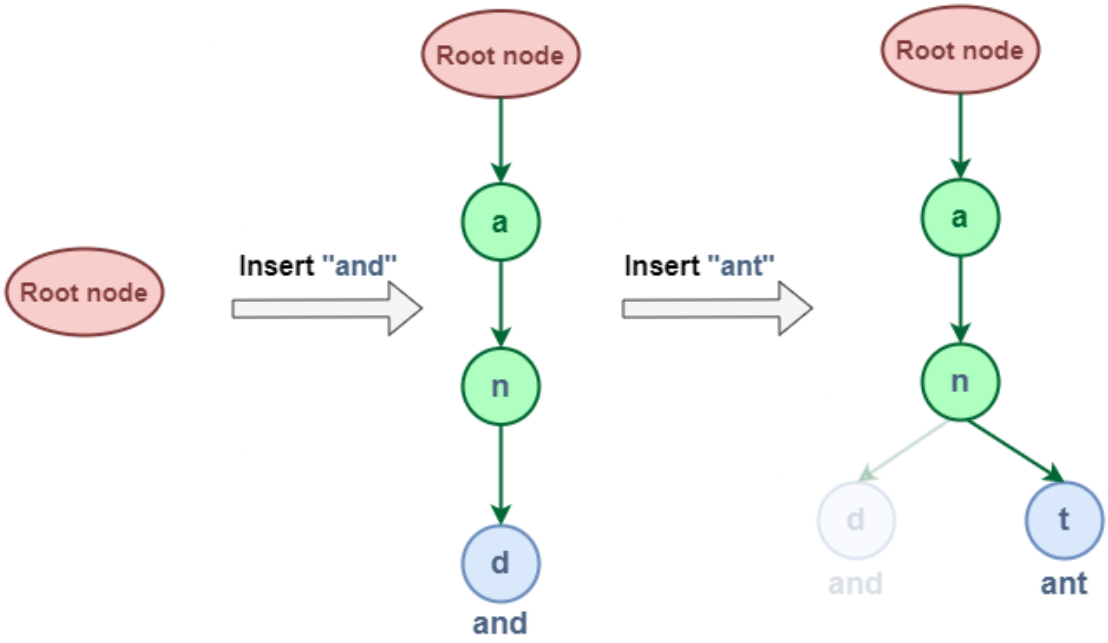
- Ký tự đầu tiên “n”: Tính toán chỉ số bằng cách sử dụng ' n' – 'a' = 13. Kiểm tra xem child [13] có phải là null không. Không phải, vì vậy hãy di chuyển đến nút “ n ” hiện có.

- Ký tự thứ hai “t”: Tính toán chỉ số bằng cách sử dụng 't' – 'a' = 19. Kiểm tra xem child [19] có phải là null không . Có, vì vậy hãy tạo một TrieNode mới với ký tự “ t ”, wordEnd được đặt thành 1 (chỉ ra từ “ant” kết thúc tại đây).

```
 // Method to insert a key into the Trie
void insert(TrieNode* root, const string& key) {

   // Initialize the curr pointer with the root node
    TrieNode* curr = root;

    // Iterate across the length of the string
    for (char c : key) {

        // Check if the node exists for the
        // current character in the Trie
        if (curr->children[c - 'a'] == nullptr) {

            // If node for current character does
            // not exist then make a new node
            TrieNode* newNode = new TrieNode();

            // Keep the reference for the newly
            // created node
            curr->children[c - 'a'] = newNode;
        }

        // Move the curr pointer to the
        // newly created node
```

```
            curr = curr->children[c - 'a'];
        }

        // Mark the end of the word
        curr->isLeaf = true;
    }
```

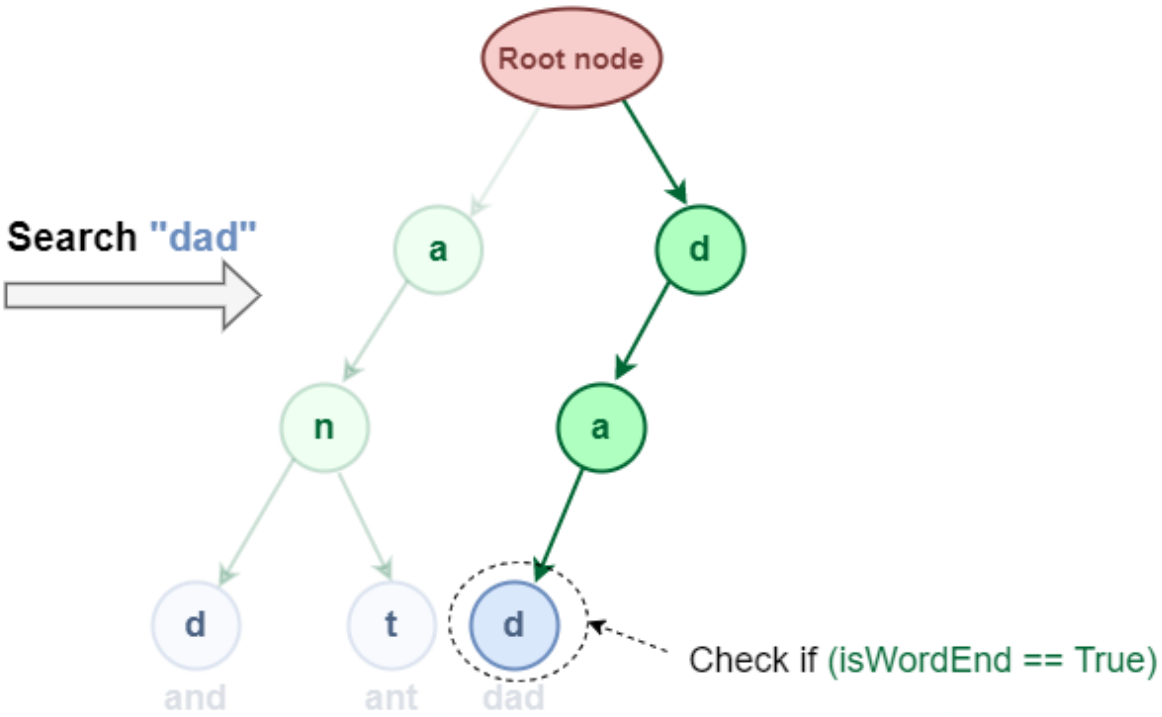
Độ phức tạp thời gian: O(n), trong đó n là độ dài của từ cần chèn
Ảnh mô phỏng chèn and và ant



- Tìm kiếm trong Cấu trúc dữ liệu Trie

    - Tìm kiếm khóa trong cấu trúc dữ liệu Trietương tự như thao tác chèn của nó. Tuy nhiên, nó chỉ **so sánh các ký tự và di chuyển xuống** . Việc tìm kiếm có thể kết thúc do kết thúc chuỗi hoặc thiếu khóa trong trie.

    - Sau đây là hình ảnh minh họa tìm kiếm từ " **dad** " trong cấu trúc dữ liệu Trie:
      Giả sử chúng ta đã chèn thành công các từ " **and** ", " **ant** ", và " **dad** " vào Triecủa chúng ta, và chúng ta phải tìm kiếm các từ cụ thể trong cấu trúc dữ liệu Trie. Hãy thử tìm kiếm từ " **dad** ":



- Chúng ta bắt đầu từ nút gốc.
- Chúng ta đi theo nhánh tương ứng với ký tự 'd'.
- Chúng ta đi theo nhánh tương ứng với ký tự 'a'.
- Chúng ta đi theo nhánh tương ứng với ký tự 'd'.
- Chúng ta đến cuối từ và cờ wordEnd là 1 .
- Điều này có nghĩa là " bố " có mặt trong Trie.
- // Method to search a key in the Trie
- bool search(TrieNode* root, const string& key)

```
// Method to search a key in the Trie
bool search(TrieNode* root, const string& key) {

    // Initialize the curr pointer with the root node
    TrieNode* curr = root;

    // Iterate across the length of the string
    for (char c : key) {

        // Check if the node exists for the
        // current character in the Trie
        if (curr->children[c - 'a'] == nullptr)
            return false;

        // Move the curr pointer to the
        // already existing node for the
        // current character
        curr = curr->children[c - 'a'];
    }
```

```
    // Return true if the word exists
    // and is marked as ending
    return curr->isLeaf;
}
```

Độ phức tạp thời gian: O(n), trong đó n là độ dài của từ cần tìm kiếm.

- **Triển khai các hoạt động chèn và tìm kiếm trong cấu trúc dữ liệu Trie**

  - Cách tiếp cận từng bước:
  - Tạo một nút gốc với sự trợ giúp của hàm tạo TrieNode() .
  - Lưu trữ một tập hợp các chuỗi cần được chèn vào Trietrong một vectơ chuỗi, chẳng hạn như arr .
  - Chèn tất cả các chuỗi vào Trievới sự trợ giúp của hàm insertKey() ,
  - Tìm kiếm chuỗi với sự trợ giúp của hàm searchKey() .

```
#include <bits/stdc++.h>
using namespace std;

class TrieNode {
public:

    // Array for children nodes of each node
    TrieNode* children[26];

    // for end of word
    bool isLeaf;

    TrieNode() {
        isLeaf = false;
        for (int i = 0; i < 26; i++) {
            children[i] = nullptr;
        }
    }
};

// Method to insert a key into the Trie
void insert(TrieNode* root, const string& key) {

    // Initialize the curr pointer with the root node
    TrieNode* curr = root;

    // Iterate across the length of the string
    for (char c : key) {

        // Check if the node exists for the
        // current character in the Trie
        if (curr->children[c - 'a'] == nullptr) {
```

```
            // If node for current character does
            // not exist then make a new node
            TrieNode* newNode = new TrieNode();

            // Keep the reference for the newly
            // created node
            curr->children[c - 'a'] = newNode;
        }

        // Move the curr pointer to the
        // newly created node
        curr = curr->children[c - 'a'];
    }

    // Mark the end of the word
    curr->isLeaf = true;
}

// Method to search a key in the Trie
bool search(TrieNode* root, const string& key) {

    if (root == nullptr) {
        return false;
    }

    // Initialize the curr pointer with the root node
    TrieNode* curr = root;

    // Iterate across the length of the string
    for (char c : key) {

        // Check if the node exists for the
        // current character in the Trie
        if (curr->children[c - 'a'] == nullptr)
            return false;

        // Move the curr pointer to the
         // already existing node for the
        // current character
        curr = curr->children[c - 'a'];
    }

    // Return true if the word exists
    // and is marked as ending
    return curr->isLeaf;
}

int main() {

    // Create am example Trie
    TrieNode* root = new TrieNode();
    vector<string> arr =
      {"and", "ant", "do", "geek", "dad", "ball"};
    for (const string& s : arr) {
```

```
        insert(root, s);
    }

    // One by one search strings
    vector<string> searchKeys = {"do", "gee", "bat"};
    for (string& s : searchKeys) {
        cout << "Key : " << s << "\n";
        if (search(root, s))
            cout << "Present\n";
        else
            cout << "Not Present\n";
    }

    return 0;
}
```

- Đầu ra

```
Từ khóa: làm
Hiện tại
Từ khóa: gee
Không có mặt
Từ khóa: dơi
Không có mặt
```

- **Phân tích độ phức tạp của cấu trúc dữ liệu Trie**

Hoạt động	Độ phức tạp thời gian
Chèn	O(n) Ở đây n là độ dài của chuỗi được chèn vào
Tìm kiếm	O(n) Ở đây n là độ dài của chuỗi được tìm kiếm

Phần 4 : Trietree cài đặt bằng con trỏ

4.1 So sánh giữa mảng và con trỏ

	Mảng (Array)	Con trỏ (Pointer)
Bản chất	Một tập hợp các phần tử có kích thước cố định, được lưu trữ liên tục trong bộ nhớ.	Biến chứa địa chỉ của một biến khác, có thể trỏ đến bất kỳ vùng nhớ nào.
Kích thước	Cố định sau khi khai báo	Có thể thay đổi bằng cấp phát động
Truy xuất phần tử	Dùng chỉ số (arr[i]). Truy cập nhanh và trực tiếp.	Dùng phép toán con trỏ (*(ptr + i)). Truy cập gián tiếp.
Bộ nhớ	Cấp phát tĩnh, bộ nhớ liên tục.	Cấp phát động hoặc tĩnh, có thể rải rác trong bộ nhớ.
Hiệu suất	Nhanh hơn, vì dữ liệu được lưu liên tục.	Có thể chậm hơn do truy xuất gián tiếp và có khả năng phân mảnh bộ nhớ
Quản lý bộ nhớ	Không cần giải phóng thủ công	Cần giải phóng bộ nhớ khi không dùng tới
Tính linh hoạt	Kích thước cố định, khó mở rộng.	Có thể mở rộng bằng cấp phát động
Tương thích hàm	Truyền nguyên mảng sẽ chuyển thành con trỏ.	Dễ truyền vào hàm, có thể thay đổi dữ liệu mà nó trỏ đến.
Cấp phát bộ nhớ động	Không thể	Có thể cấp phát và giải phóng động.
Ứng dụng	Lưu trữ dữ liệu có kích thước cố định như danh sách số nguyên, chuỗi ký tự.	Quản lý bộ nhớ linh hoạt, cấu trúc dữ liệu động như danh sách liên kết, cây nhị phân.

Note :

- Dùng mảng khi kích thước dữ liệu cố định và cần truy xuất nhanh.
- Dùng con trỏ khi cần quản lý bộ nhớ động hoặc làm việc với cấu trúc dữ liệu phức tạp như danh sách liên kết, cây, đồ thị.

4.2 Mô phỏng code

```
// Method to search a key in the Trie
#include <bits/stdc++.h>
using namespace std;
struct node {
    node* child[26];
    bool isEnd;

       node() {
        memset(child, NULL, sizeof(child));
        isEnd = 0;
    }
};

node* root = new node();

void Add(string &s) {
       node* u = root;
    for (int i = 0; i < s.size(); ++i) {
           int k = s[i] - 'a';
           if (u->child[k] == NULL)
               u->child[k] = new node();
           u = u->child[k];
       }
    u->isEnd = 1;
   }

bool Query(string &s) {
    node* u = root;
    for (int i = 0; i < s.size(); ++i) {
        int k = s[i] - 'a';
        if (u->child[k] == NULL) return 0;
        u = u->child[k];
    }
    return u->isEnd;
}

int main() {
    vector<string> words;
    int n,m;
    cin>>n>>m;
    for(int i=0;i<n;i++){
        string temp;
        cin>>temp;
        words.push_back(temp);
    }

    for (auto &word : words) {
```

```
        Add(word);
    }
    vector<string> queries;
    for(int i=0;i<m;i++){
        string temp;
     cin>>temp;
           queries.push_back(temp);

    }

    for (auto &q : queries) {
        if (Query(q)) {
               cout << q <<" YES"<<endl;
        } else {
            cout << q <<" NO"<<endl;
     }
    }

    return 0;
}
```

- Hết -