Cấu trúc dữ liệu và giải thuật
Báo cáo tìm hiểu cấu trúc dữ liệu Trie

Thông tin nhóm thực hiện

Họ tên	MSSV
Đoàn Gia Huệ	22120116
Phạm Gia Khang	22120152
Phạm Gia Khiêm	22120159
Lê Anh Khôi	22120165

1 Độ phức tạp thuật toán

```
struct TrieNode
{
    TrieNode *child[26] = {nullptr};
    bool isWord = false;
};
```

1.1 Thêm một từ vào Trie

```
void addWord(TrieNode *&root, const string &word)
{
    TrieNode *i = root;
    for (const char &x : word)
    {
        if (!i->child[x - 'a'])
            i->child[x - 'a'] = new TrieNode;
        i = i->child[x - 'a'];
    }
    i->isWord = true;
}
```

Do việc thêm một từ vào Trie chỉ cần đi qua từng kí tự của từ và thêm tuần tự vào Trie nên độ phức tạp thuật toán thêm một từ vào Trie là $O(n)$ với n là độ dài của từ cần thêm vào.

1.2 Xóa một từ khỏi Trie

```
bool isLeaf(TrieNode *root)
{
    for (int i = 0; i < 26; ++i)
        if (root->child[i])
            return false;
    return true;
}

TrieNode *deleteWord(TrieNode *&root,
                     const string &word,
                     int index)
{
    if (!root)
        return nullptr;

    if (index == word.size())
    {
        root->isWord = false;
        if (isLeaf(root))
            delete root, root = nullptr;
        return root;
    }

    if (root->child[word[index]])
        root = deleteWord(root, word, index + 1);

    if (isLeaf(root) && !root->isWord)
        delete root, root = nullptr;
    return root;
}
```

Để xóa một từ trong Trie, đầu tiên ta sẽ tìm đến node đánh dấu vị trí kết thúc của từ đó trong Trie, sau đó ta sẽ đệ quy ngược lại để xóa các node lá mà không phải là từ trong Trie. Việc tìm có thời gian là $O(n)$ và việc xóa có thời gian tối đa là $O(n)$. Vậy độ phức tạp thời gian của phép xóa từ là $O(n)$ với n là độ dài của từ cần xóa.

1.3 Tìm một từ trong Trie

```
bool search(TrieNode* root, const string& word) {
    TrieNode* i = root;
    for (const char& x : word) {
        if (!i->child[x - 'a'])
            return false;
        i = i->child[x - 'a'];
    }
    return i->exist;
}
```

Do việc tìm một từ trong Trie chỉ cần đi qua từng kí tự của từ và kiểm tra kí tự đó có tồn tại trong Trie hay không nên độ phức tạp của thuật toán tìm một từ trong Trie là tương tự như thêm một từ vào Trie là $O(n)$.

1.4 Tìm từ cùng tiền tố có độ dài i trong Trie

```
void searchWordWithLength(TrieNode *root,
                          int length,
                          string &cur,
                          vector<string> &result)
{
    if (length == 0 && root->isWord)
        result.push_back(cur);
    else
    {
        for (int i = 0; i < 26; ++i)
        {
            if (root->child[i])
            {
                cur += (char)(i + 'a');
                searchWordWithLength(root->child[i],
                                     length - 1,
                                     cur,
                                     result);
                cur.pop_back();
            }
        }
    }
}
```

```
vector<string> searchWordsWithSamePrefixOfLength
                        (TrieNode *root,
                        const string &prefix,
                        int length)
{
    TrieNode *i = root;

    if (length < prefix.size())
        return {};

    int neededLength = length - prefix.size();

    for (const char &x : prefix)
    {
        if (!i->child[x - 'a'])
            return {};
        i = i->child[x - 'a'];
    }

    vector<string> result;
    string temp = "";
    searchWordWithLength(i, neededLength, temp, result);

    for (string &x : result)
        x = prefix + x;
    return result;
}
```

Việc tìm từ có cùng tiền tố với độ dài i được tách ra làm 2 bước. Bước đầu tiên là đi tìm tiền tố đó, bước này sẽ tốn thời gian là $O(m)$ với m là độ dài của tiền tố. Bước tiếp theo là tìm tất cả các từ có độ dài $i - m$, bước này sẽ tốn khá nhiều thời gian bởi số từ thỏa mãn có thể rất lớn, độ phức tạp của bước này là $O(M)$ với M là số từ thỏa giá trị của M có thể lên đến 26^{i-m}. Vậy độ phức tạp thời của thuật toán này là $O(M + m)$.

2 Những điểm mạnh của Trie so với các cấu trúc dữ liệu khác

2.1 Cây tìm kiếm nhị phân

- Việc tìm một từ trong Trie sẽ nhanh hơn nhiều so với Cây tìm kiếm nhị phân. Độ phức tạp để tìm/thêm một từ trong Trie là $O(l)$ với l là độ dài của từ cần thêm, trong khi đó Cây nhị phân tìm kiếm cần đến $O(\log n)$ với n là số node hiện có trong cây trong truòng hợp cây cân bằng.

2.2 Bảng băm

- So với bảng băm, có thể việc kiểm tra chính xác một từ thì Trie không nhanh bằng, nhưng việc tìm kiếm từ theo một số chữ cái hoặc một tiền tố nhất định thì bảng băm lại làm không tốt bằng Trie. Nguyên nhân phần lớn nằm ở tính chất của Trie, chính là lưu trữ các từ vựng theo từng tiền tố.
- Việc thêm một từ vào Trie sẽ tốn ít thời gian hơn, bởi Trie chỉ cần tốn n bước để thêm một từ vào, trong khi đó việc thêm một từ vào bảng băm cần ít nhất $c \times n$ bước để thực hiện với hằng số c sẽ lớn hơn nhiều so với 1.