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 Phân tích độ phức tạp thuật toán

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

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 một từ có cùng tiền tố độ dài i trong Trie

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

2.1 Cây tìm kiếm nhị phân

Bản chất Trie cũng là một cây tìm kiếm, thế nhưng trong trường hợp lưu các từ vựng, Trie tỏ ra vượt trội và tiện hơn hẳn một cây tìm kiếm nhị phân. Bởi khi này, Trie là một cây tìm kiếm 26-phân, nên việc tìm kiếm các từ vựng theo lý thuyết sẽ tiện lợi và nhanh hơn.

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.