Trie Auto-complete App

23120124 – Nguyễn Minh Hiếu
23120124@student.hcmus.edu.vn

Nội dung trình bày

I. Giới thiệu
II. Các thao tác trên Compressed Trie
III. Thực nghiệm
IV. Minh họa ứng dụng
V. Tài liệu tham khảo

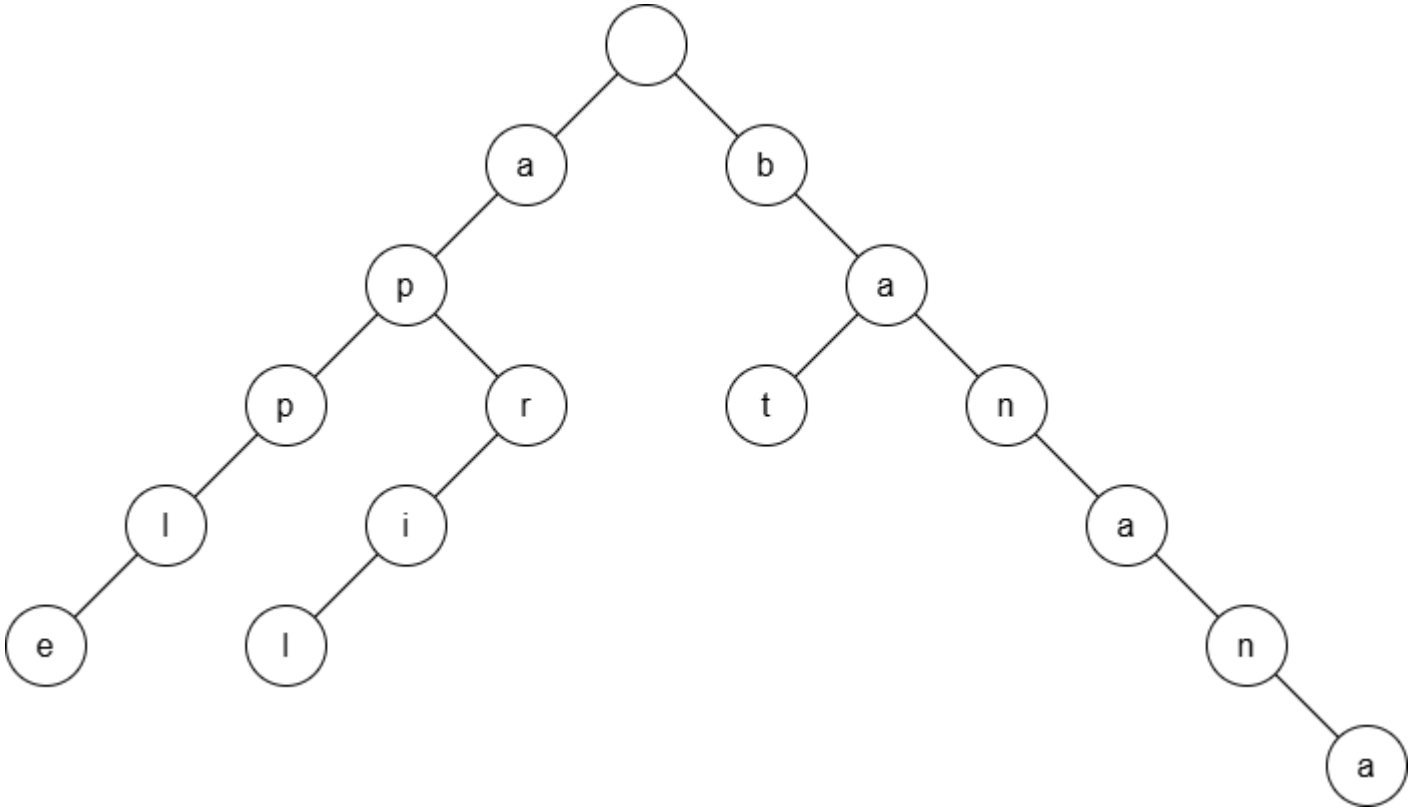
I. Giới thiệu

1. Standard Trie

- Trie (hay còn gọi là cây tìm kiếm tiền tố) là một cấu trúc dữ liệu dạng cây dùng để lưu trữ các chuỗi ký tự.
- Mỗi nút trong cây Trie biểu diễn một ký tự ngoại trừ nút gốc và các con trỏ tới các nút con
- Mỗi đường dẫn từ gốc đến bất kỳ nút nào đều biểu diễn một từ hoặc chuỗi.
- Ví dụ: cấu trúc của một nút trong Trie

```
1  struct TrieNode
2      bool isEnd
3      unordered_map <char, TrieNode*> child
```

- **Ví dụ**: cây trie cho các chuỗi app, apple, april, bat, banana



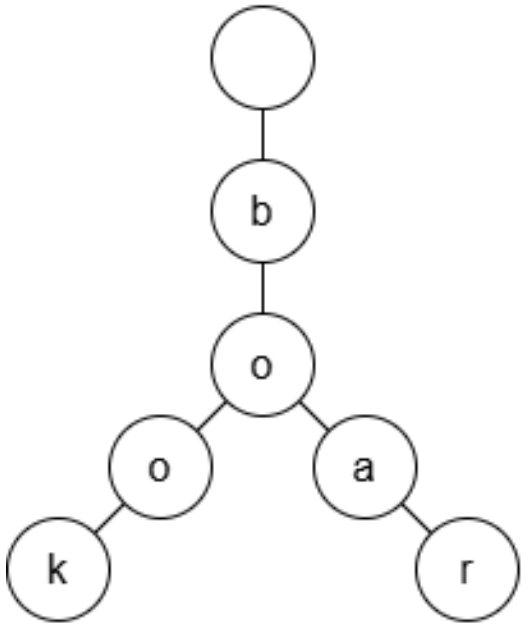
I. Giới thiệu

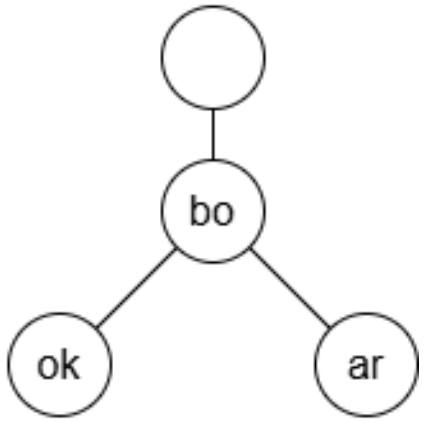
2. Compressed Trie

- Được xây dựng bằng cách nén các nút "dư thừa" của Standard Trie để đạt được tối ưu hóa không gian
- Về mặt bộ nhớ, Compressed Trie sử dụng rất ít nút, mang lại lợi thế lớn về bộ nhớ (đặc biệt là đối với các chuỗi dài) có tiền tố chung dài.
- Về mặt tốc độ, cây trie thông thường sẽ nhanh hơn một chút vì các hoạt động của nó không liên quan đến bất kỳ hoạt động chuỗi nào, chúng là các vòng lặp đơn giản.
- Ví dụ cấu trúc nút của Compressed Trie

```
1  struct TrieNode
2      string key
3      bool isEnd
4      unordered_map <char, TrieNode*> child
```

- **Ví dụ**: cây bên trái là Trie chuẩn, cây bên phải là Trie nén

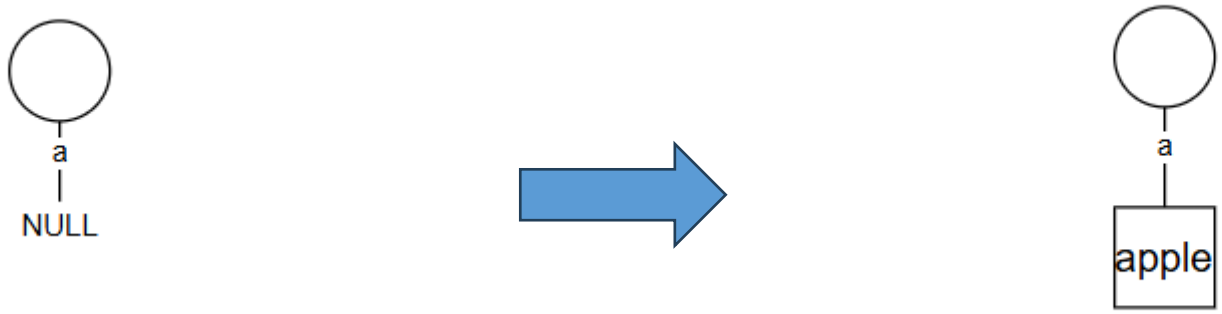




I. Các thao tác trên Compressed Trie

1. Insert

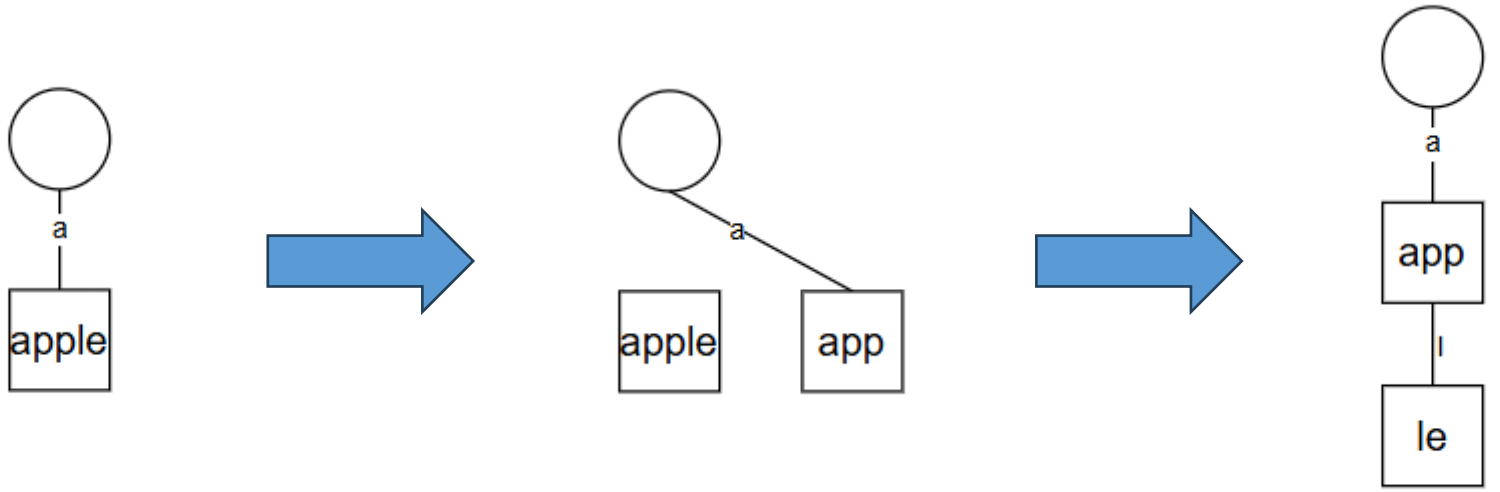
- Ví dụ: Thêm lần lượt các từ apple, app, april, apples
  - Thêm apple:



II. Các thao tác trên Compressed Trie

1. Insert

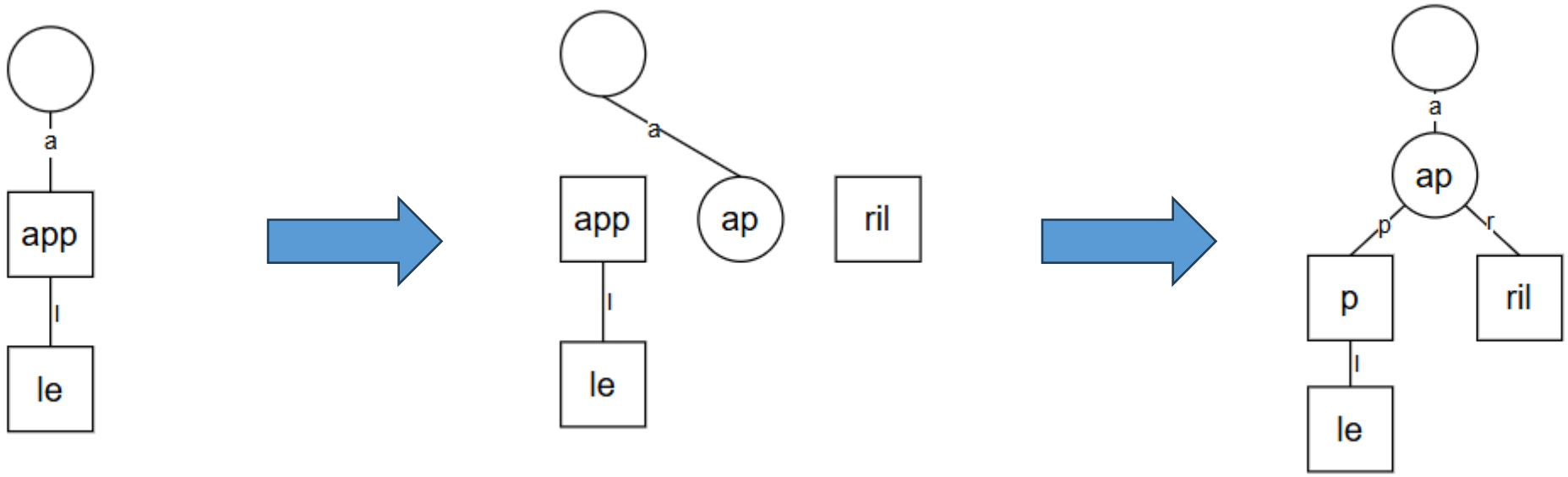
- Thêm app:



II. Các thao tác trên Compressed Trie

1. Insert

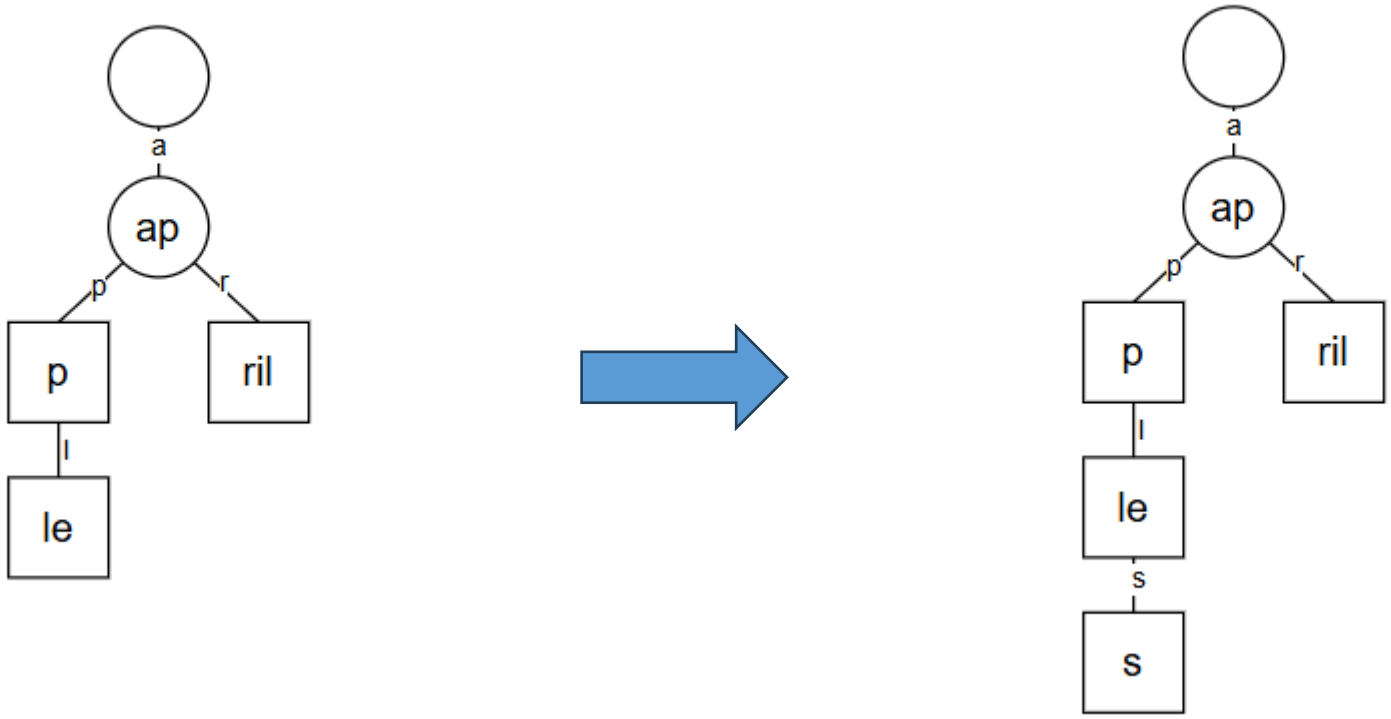
- Chèn april:



II. Các thao tác trên Compressed Trie

1. Insert

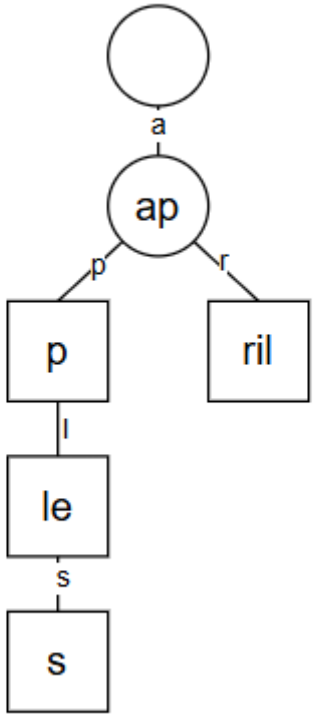
- Ví dụ: Thêm apples



II. Các thao tác trên Compressed Trie

2. Find

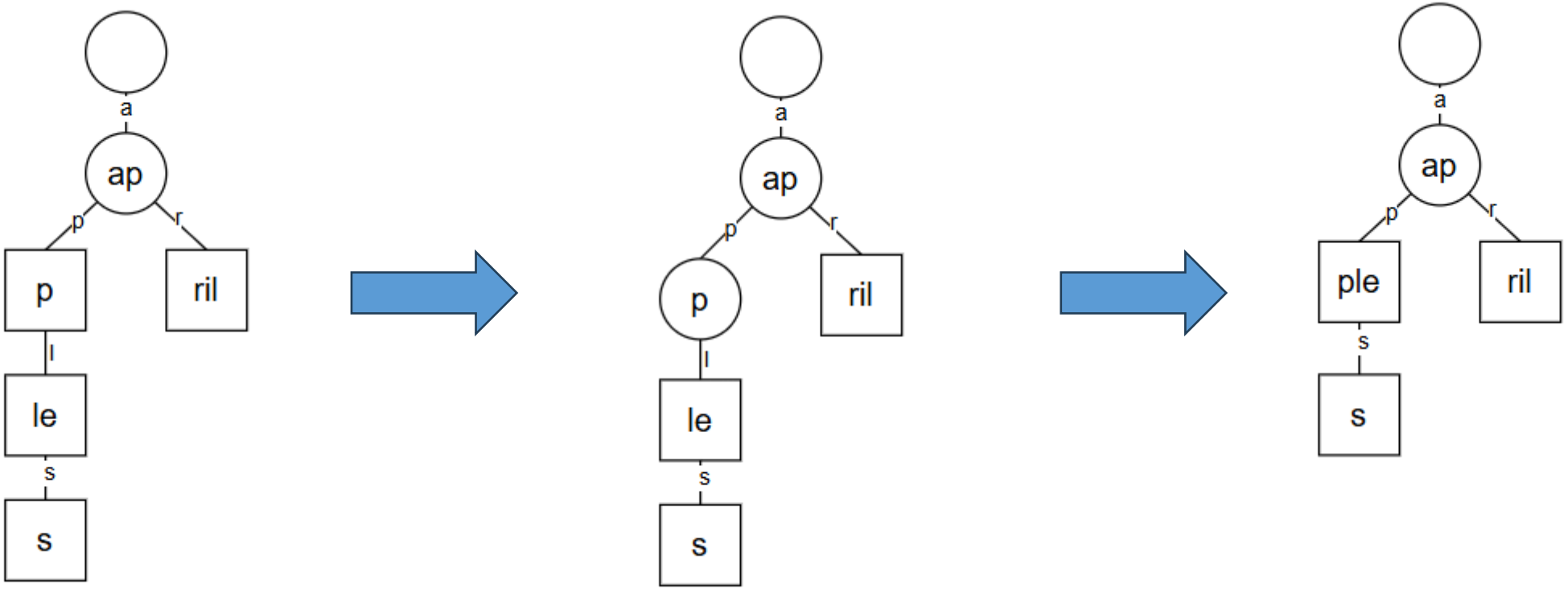
- Ví dụ: Từ apple có trong cây sau không?



II. Các thao tác trên Compressed Trie

3. Remove

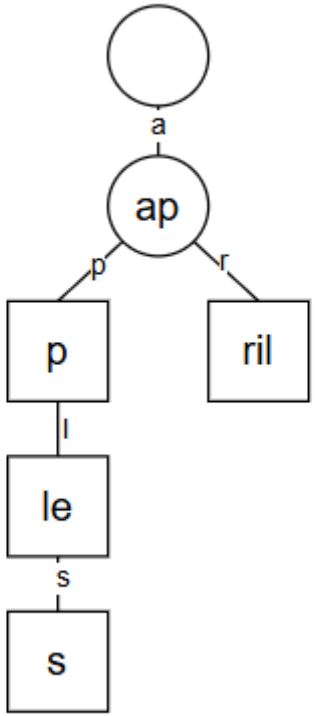
- Ví dụ: Xóa từ app khỏi cây



II. Các thao tác trên Compressed Trie

4. Auto-complete

- Ví dụ: Liệt kê tất cả các từ bắt đầu bởi “ap” trong cây sau



II. Các thao tác trên Compressed Trie

5. Đánh giá độ phức tạp

- Độ phức tạp bộ nhớ: O(n)
- Độ phức tạp thời gian trung bình:
  - Insert: O(m)
  - Find: O(m)
  - Remove: O(m)
  - Auto-complete: O(m + k*L)
- Trong đó:
  - n: tổng số ký tự của các chuỗi
  - m: số ký tự của chuỗi người dùng nhập
  - K: số lượng từ gợi ý
  - L: số lượng ký tự còn lại của chuỗi có chung tiền tố

III. Thực nghiệm

1. Bộ dữ liệu sử dụng

- Bộ dữ liệu sử dụng trong thí nghiệm là từ điển tiếng Anh lấy từ DWYL English Words GitHub Repository
- File “word.txt” danh sách hơn 479000 từ tiếng Anh, mỗi từ được tạo thành từ các chữ cái Latin, hoặc các chữ số và một sô các ký tự đặc biệt khác

III. Thực nghiệm

1. Bộ dữ liệu sử dụng

- Một thuật toán khác cho bài toán auto-complete: Binary Search
- Đánh giá độ phức tạp thời gian O(N + K)
- Trong đó:
  - N: tổng số lượng từ trong từ điển
  - K: số lượng từ gợi ý

III. Thực nghiệm

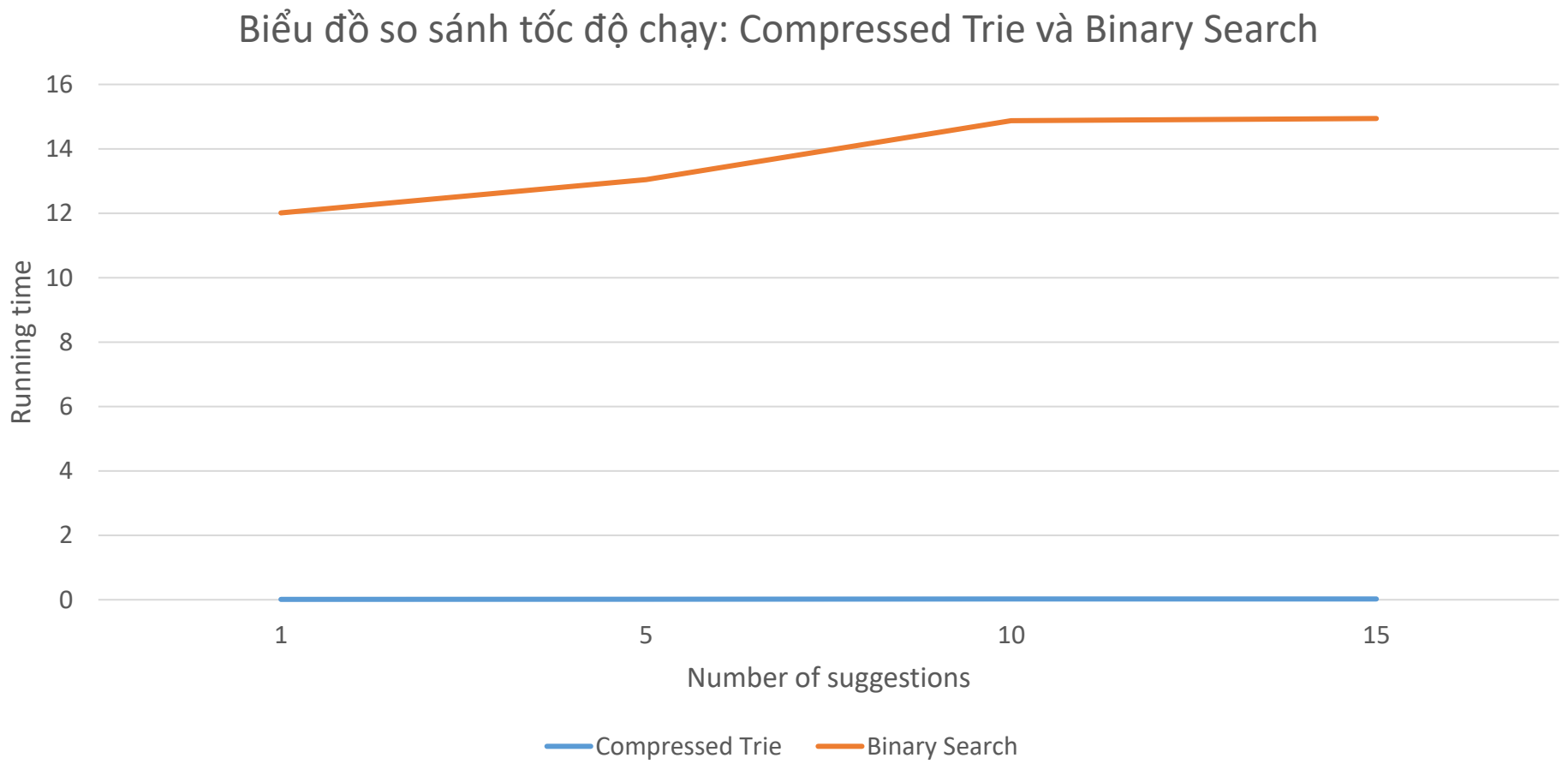
2. Kết quả

Number of suggestions	1		5		10		15	
Resulting statics	Running time (ms)	Comparison	Running time (ms)	Comparison	Running time (ms)	Comparison	Running time (ms)	Comparison
Compressed Trie	0.0111	33	0.012	75	0.0214	91	0.0242	117
Binary Search	12.0118	54	13.0449	82	14.8721	117	14.9423	152

Bảng: thời gian chạy và số lượng phép so sánh của Compressed trie và Binary search trong bài toán auto-complete

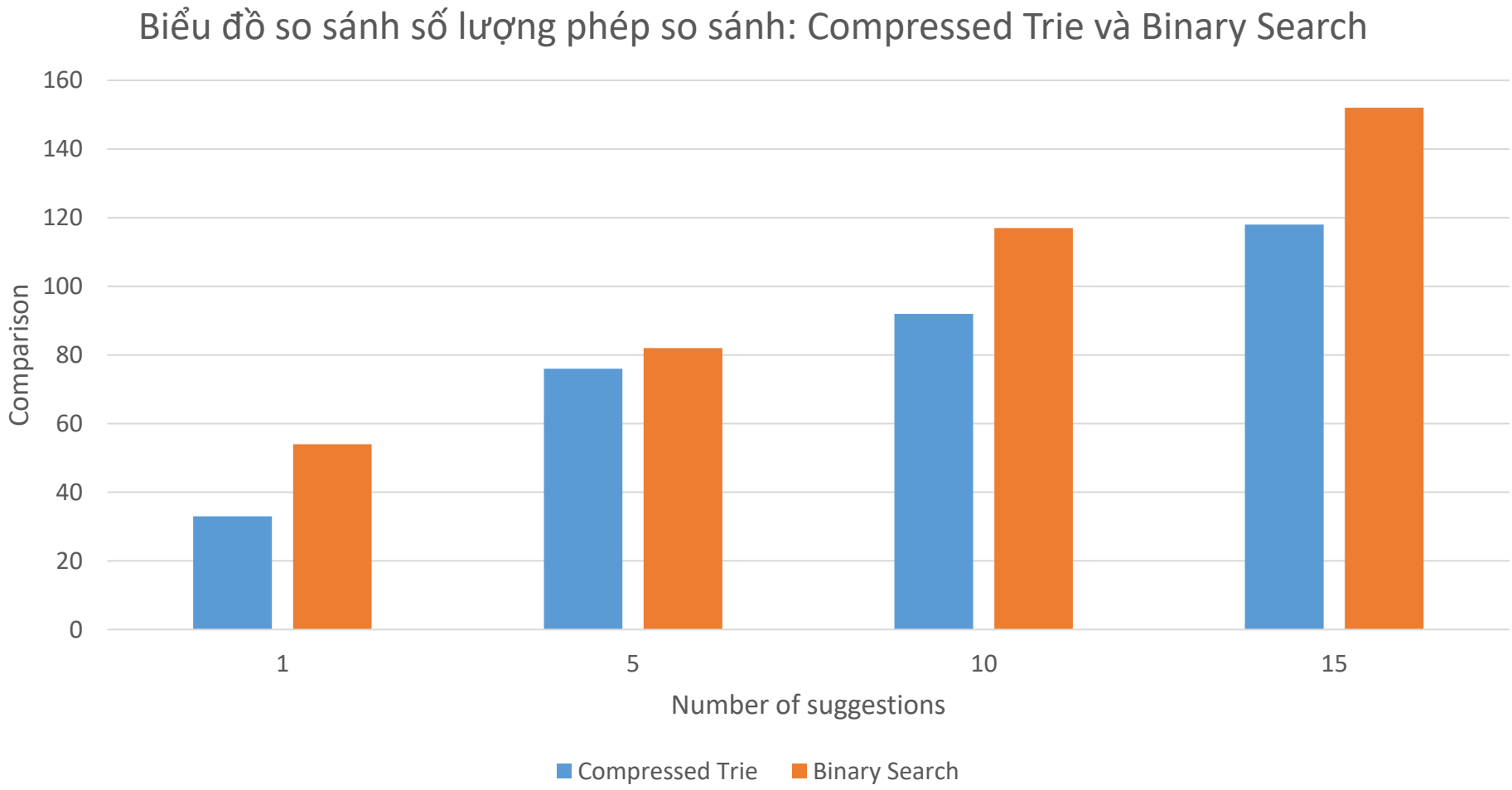
III. Thực nghiệm

2. Kết quả



III. Thực nghiệm

2. Kết quả



III. Thực nghiệm

3. Nhận xét & kết luận

- Binary Search
  - Nhanh trong việc tìm kiếm một từ cụ thể nếu danh sách đã được sắp xếp.
  - Tuy nhiên, độ phức tạm thời gian của Binary search phụ thuộc vào số lượng từ trong từ trong từ điển vì vậy với từ điển quá lớn thì tìm kiếm nhị phân sẽ không hiệu quả bằng Trie.
  - Đồng thời, khi từ điển chưa được sắp xếp ta sẽ tốn thêm thời gian để sắp xếp.
- Khi nào thì sử dụng Binary Search?
  - Khi từ điển có kích thước nhỏ, và đã được sắp xếp
  - Bộ nhớ hạn chế

III. Thực nghiệm

3. Nhận xét & kết luận

- Compressed Trie
  - Có thể tìm kiếm nhanh chóng và liệt kê các từ có chung tiền tố một cách hiệu quả.
  - Đặc biệt, khi từ điển lớn, độ phức tạp của Compressed Trie phụ thuộc vào số lượng ký tự trong tiền tố (prefix) và tổng số lượng ký tự còn lại của các từ có chung tiền tố, chứ không phụ thuộc vào tổng số lượng từ trong từ điển. Điều này khiến Compressed Trie trở thành lựa chọn tối ưu khi số lượng từ rất lớn
- Khi nào thì sử dụng Compressed Trie?
  - Số lượng từ lớn và có nhiều tiền tố chung
  - Khi từ cần thực hiện các thao tác thêm và xóa trên từ điển

IV. Minh họa ứng dụng

```
----------=====*******=====----------

                -*  SEARCH  *-

                    INSERT

                    REMOVE

----------=====*******=====----------
```

IV. Tài liệu tham khảo

[1]. Adam Drozdek, Data structure and algorithm in c++, Fourth Edition

[2]. Goodrich, Tamassia 2004, Trie