BÁO CÁO CHALLENGE 1

Cây Trie có một số ưu điểm so với các cấu trúc dữ liệu khác như Cây Tìm kiếm Nhị phân (Binary Search Tree - BST) và Hash Table khi được thiết kế cho mục đích tìm kiếm:

1. Tìm kiếm hiệu quả:

- Trie thường cung cấp thời gian tìm kiếm hiệu quả hơn so với Cây Tìm kiếm Nhị phân và Bảng Hash trong một số trường hợp. Đặc biệt là khi tìm kiếm các từ hoặc chuỗi có chung tiền tố.

Ví dụ:

BST: Nếu cây BST được tạo theo thứ tự tăng dần hoặc giảm dần, thì thời gian tìm kiếm có thể trở nên độ phức tạp cao (O(n)).

Hash Table: Trong trường hợp xung đột, việc tìm kiếm có thể trở nên chậm.

2. Tìm kiếm theo tiền tố (Prefix Search):

- Trie làm cho việc tìm kiếm các từ bắt đầu bằng một tiền tố cụ thể trở nên đơn giản và hiệu quả. Các node chung của các từ có cùng tiền tố sẽ được chia sẻ, giảm đáng kể thời gian tìm kiếm.

Ví dụ:

Trie: Nếu có nhiều từ có chung tiền tố (ví dụ: "apple" và "appetizer"), Trie có thể tìm kiếm nhanh chóng với độ phức tạp thời gian O(l), trong đó l là độ dài của từ cần tìm kiếm.

3. Không Gian Lưu Trữ Hiệu Quả:

- Trong một số trường hợp, Trie có thể tiết kiệm không gian lưu trữ so với BST và Hash table. Điều này đặc biệt quan trọng khi lưu trữ một lượng lớn các từ có sự chia sẻ nhiều tiền tố.

Ví dụ:

BST: Cần lưu trữ thêm thông tin về mỗi node (thường cần 2 con trỏ), có thể dẫn đến lãng phí không gian.

Hash Table: Có thể dẫn đến lãng phí không gian khi bảng phải được mở rộng.

4. Khả năng Thêm và Xóa Hiệu Quả:

- Các thao tác thêm và xóa trên Trie thường hiệu quả hơn so với BST, đặc biệt là khi thêm hoặc xóa các từ có sự chia sẻ nhiều tiền tố. Trong khi Hash table cũng có khả năng thêm và xóa tốt, nhưng nó có thể phải xử lý các xung đột.

Ví dụ:

BST: Cần cân nhắc tái cấu trúc cây nếu dữ liệu được thêm theo thứ tự gần tăng dần hoặc giảm dần.

Trie: Thêm và xóa có thể được thực hiện một cách hiệu quả với độ phức tạp thời gian O(k), trong đó k là độ dài của từ.

5. Khả năng Tự Động Mở Rộng:

- Trie có thể mở rộng tự động cho bất kỳ tập dữ liệu nào mà không cần tái cấu trúc. Điều này làm cho Trie linh hoạt khi tập dữ liệu thay đổi động.

Ví dụ:

Trie: Có thể mở rộng mà không cần tái cấu trúc. Nếu thêm từ mới, chỉ cần thêm các node mới vào Trie.

Hash Table: Có thể cần tái cấu trúc khi có nhiều xung đột, điều này có thể gây lãng phí thời gian.