트라이(Trie) 자료구조의 기본 개념과 예시

practice 자료구조 · 2024. 10. 14. 13:53

Q. 트라이(Trie) 자료구조의 기본 개념과 예시는 구체적으로 어떤건가요

트라이는 **문자열 검색**에 특화된 트리 구조로, 주로 자동 완성, 접두사 검색, 사전 구현에 사용됩니다. 각 노드는 하나의 문자를 저장하며, 문자열은 루트에서부터 각 문자를 따라가면서 표현됩니다. 접두사를 공유하므로 문자열 탐색이 빠르며, 삽입 및 탐색 시간 복잡도는 문자열 길이에 비례해 O(L)입니다.

주요 특징
- **빠른 문자열 탐색**과 삽입
- **공간 효율성**: 접두사를 공유하여 중복을 줄임
- **활용 예**: 자동 완성 시스템, 사전 검색, 중복 문자열 검사

트라이(Trie)는 주로 **문자열 검색**을 효율적으로 처리하기 위한 트리 구조로, 문자열을 저장하고 탐색하는 데 특화된 자료구조입니다. 트라이는 주로 **사전**과 같은 데이터 구조에서 사용되며, 특히 문자열 집합에서 빠른 탐색, 자동 완성, 접두사 검색과 같은 작업을 수행할 때 매우 유용합니다.

트라이의 주요 개념

1. **노드**: 트라이의 각 노드는 문자열의 문자 하나를 저장합니다.
2. **루트 노드**: 트라이의 최상위 노드는 비어 있으며, 각 자식 노드는 문자열의 첫 글자를 저

장합니다.
3. **자식 노드**: 각 노드는 여러 자식 노드를 가질 수 있으며, 자식 노드는 현재 노드의 문자에 이어지는 문자열의 다음 문자를 저장합니다.
4. **문자열 끝 표시**: 특정 노드에서 하나의 문자열이 끝나는 지점을 표시하기 위해, 일반적으로 "종료 표시" 또는 불리언 값이 사용됩니다.

트라이의 동작

트라이에서 각 문자열은 루트에서부터 시작해, 문자열의 각 문자를 하나씩 따라가며 노드를 통해 표현됩니다. 만약 탐색 중에 이미 존재하는 문자가 있다면 그 노드를 계속해서 따라가고, 그렇지 않다면 새로운 노드를 만들어 추가합니다.

트라이의 특징

- **삽입 시간 복잡도**: \(O(L)\) (L은 삽입하려는 문자열의 길이)
- **탐색 시간 복잡도**: \(O(L)\) (L은 탐색하려는 문자열의 길이)
- **공간 복잡도**: 트라이의 공간 복잡도는 저장하는 문자열의 수와 각 문자열의 길이에 비례합니다. 최악의 경우, 트라이의 노드 수는 문자열의 개수와 각 문자열의 문자 수에 따라 많아질 수 있습니다.

트라이의 예시

다음은 트라이에 문자열을 삽입하고 탐색하는 간단한 예시입니다. 예를 들어, 문자열 집합 `["car", "cat", "dog"]`를 트라이에 삽입하는 과정을 살펴보겠습니다.

트라이 구조
1. **삽입**:
   - "car"를 삽입: `c -> a -> r` 경로가 생깁니다.
   - "cat"을 삽입: `c -> a`까지는 이미 존재하므로, `t`만 추가됩니다.
   - "dog"를 삽입: 새로운 경로 `d -> o -> g`가 추가됩니다.

트라이의 모습은 다음과 같습니다:

```
        (root)
        /    \
       c      d
```

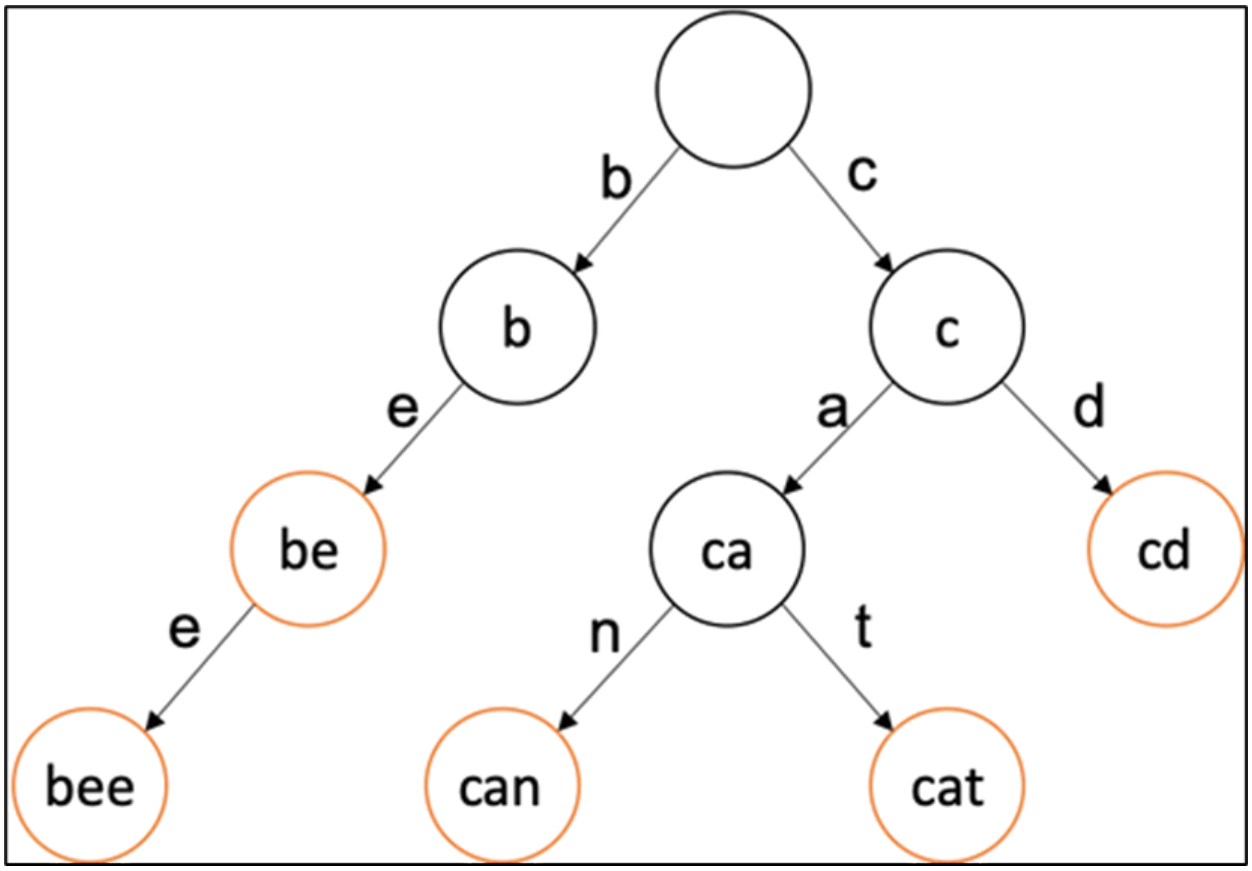
```
      /        \
     a          o
    / \          \
   r   t          g
```

```

2. **탐색**:
   - "car"를 찾을 때는 `c -> a -> r` 경로를 따라가면 됩니다.
   - "cat"을 찾을 때는 `c -> a -> t` 경로를 따라가면 됩니다.
   - "do"는 존재하지 않는 부분 문자열이므로, `dog`의 경로를 따라가다가 찾지 못합니다.

트라이는 이렇게 문자열의 접두사를 공유하여 저장하므로, 문자열 검색 및 관련 작업에서 매우 효율적입니다.

### 트라이의 활용 예

1. **자동 완성 시스템**: 사용자가 입력한 문자열의 접두사에 해당하는 단어들을 빠르게 찾아서 추천하는 기능.
2. **사전 검색**: 주어진 문자열이 사전에 존재하는지를 빠르게 확인할 수 있습니다.
3. **문자열 집합에서의 중복 검사**: 입력된 문자열이 집합에 이미 존재하는지를 효율적으로 확인할 수 있습니다.

이러한 특징 덕분에 트라이는 많은 텍스트 기반의 애플리케이션에서 널리 사용됩니다.

•다양한 문자열을 저장하고, 효율적으로 삽입, 삭제, 검색할 수 있음

•영어에는 26개의 글자가 있음 -> 최대 26진 트리 (알파벳갯수때문)

•새로운 문자열을 삽입하려면, Root부터 시작해, 매 글자마다 길을 따라간다. 이후 Leaf 노드를 만든다

•주어진 문자열을 삭제하는 것도 동일
```

•문자열 검색 또한 길을 따라가서 있으면 찾은 것

•다만 트라이는 상대적으로 많은 메모리를 필요로 함

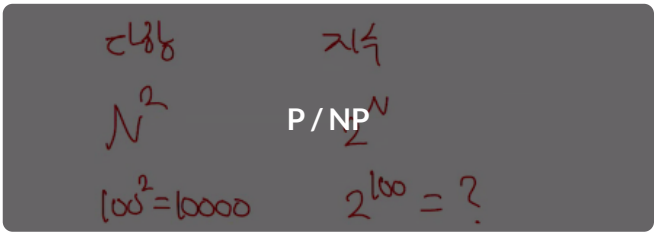
•직관적으로, 문자열을 그대로 저장하는 것과 비교했을때 더 많은 저장 공간 필요

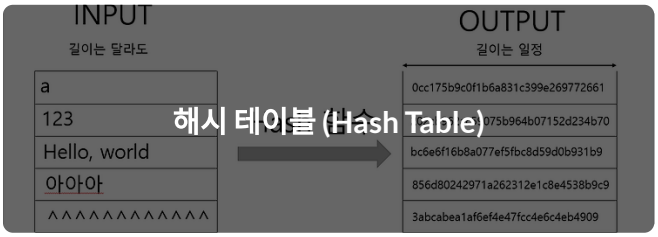


'practice_자료구조' 카테고리의 다른 글

사이클 탐지 (0) 2024.10.14
그래프 탐색- DFS,BFS (0) 2024.10.14
P / NP (0) 2024.09.30
해시 테이블 (Hash Table) (0) 2024.09.26
crossentropy (1) 2024.09.20

관련글 관련글 더보기





자연어(NLP)

네이쳐2024 님의 블로그입니다.

구독하기 +

댓글 0

이름

비밀번호

내용을 입력하세요.

등록