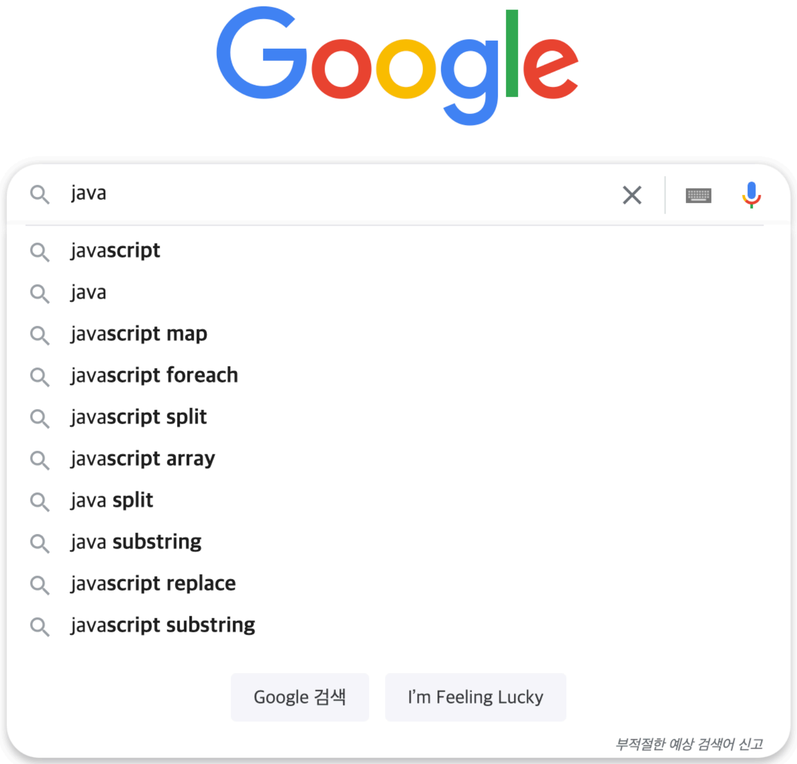
programmers

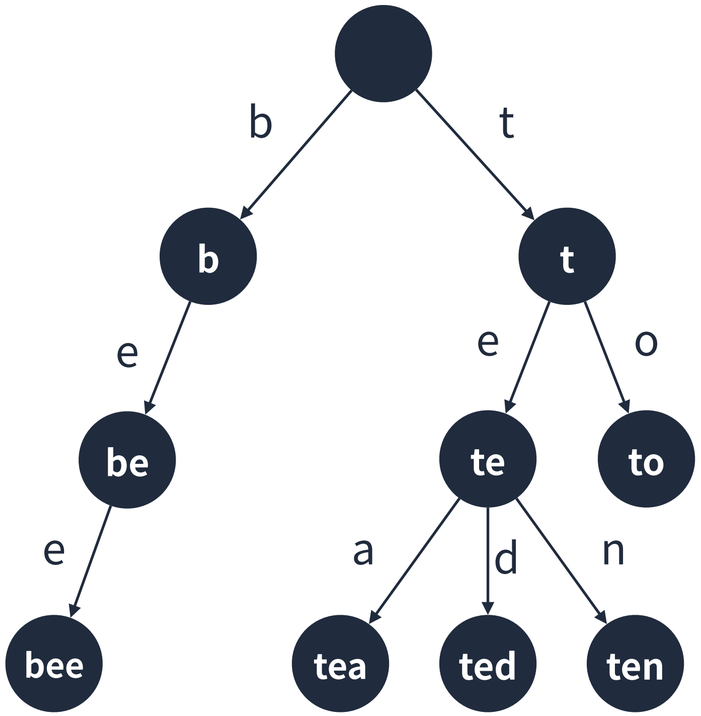
트라이

코딩테스트 광탈방지 A to Z : JavaScript - 이선협 @kciter


Google
java
javascript
java
javascript map
javascript foreach
javascript split
javascript array
java split
java substring
javascript replace
javascript substring
Google 검색
I'm Feeling Lucky
부적절한 예상 검색어 신고

트라이

문자열을 저장하고 효율적으로 탐색하기 위한 트리 형태의 자료구조



트라이의 특징

- 검색어 자동완성, 사전 찾기 등에 응용될 수 있다.
- 문자열을 탐색할 때 단순하게 비교하는 것보다 효율적으로 찾을 수 있다.
- L이 문자열의 길이일 때 탐색, 삽입은 O(L)만큼 걸린다.
- 대신 각 정점이 자식에 대한 링크를 전부 가지고 있기에 저장 공간을 더 많이 사용한다.

Trie 생성하기

트라이 구조

- 루트는 비어있다.
- 각 간선(링크)은 추가될 문자를 키로 가진다.
- 각 정점은 이전 정점의 값 + 간선의 키를 값으로 가진다.
- 해시 테이블과 연결 리스트를 이용하여 구현할 수 있다.

Step 1

Root

처음엔 빈 루트 정점만 존재한다.

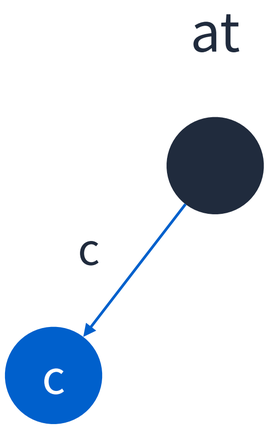
Step 1

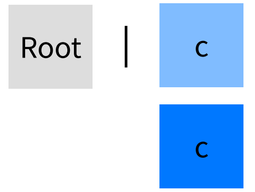
cat

Root

여기에 cat을 추가해보자

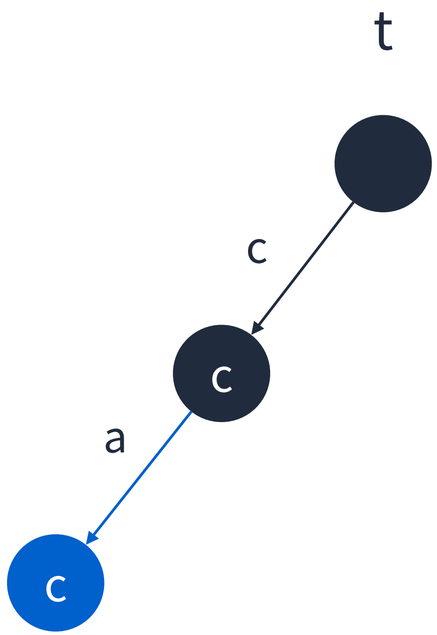
Step 2

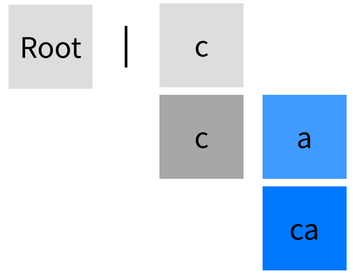




맨 앞 문자열 `c`를 잘라 루트 정점의 자식으로 추가한다

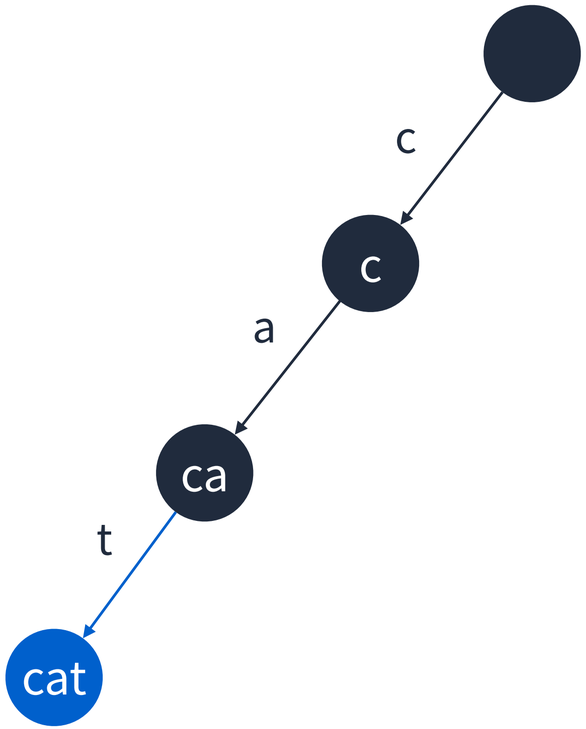
Step 3

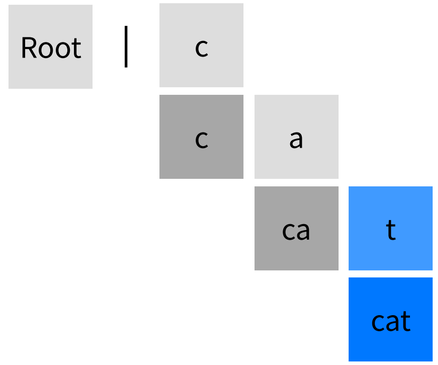




맨 앞 문자열 `a`를 잘라 c 정점의 자식으로 추가한다

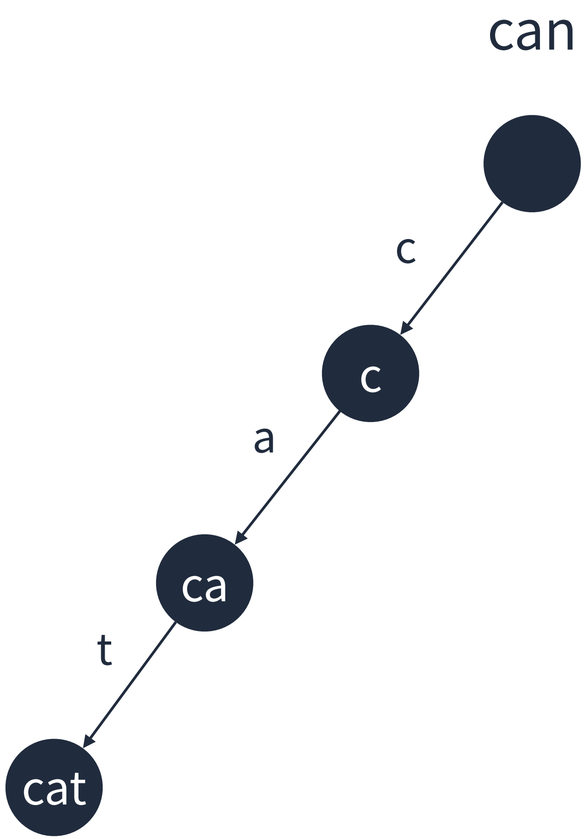
Step 4

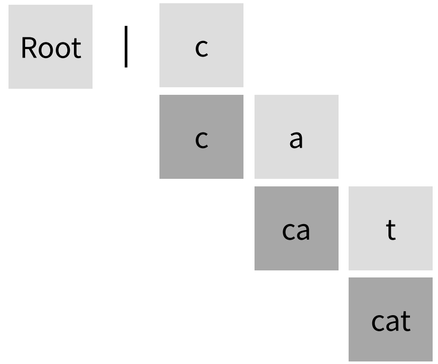




마지막 문자열 `t`를 잘라 ca 정점의 자식으로 추가한다

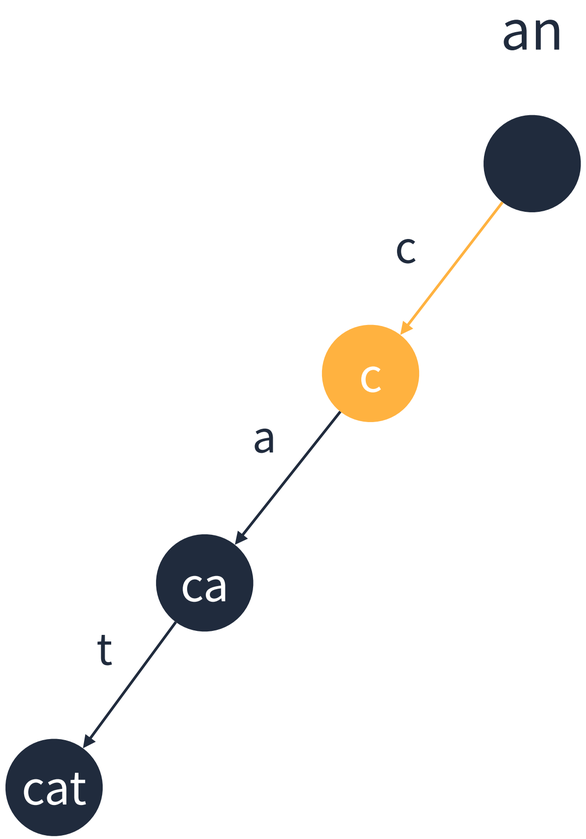
Step 5

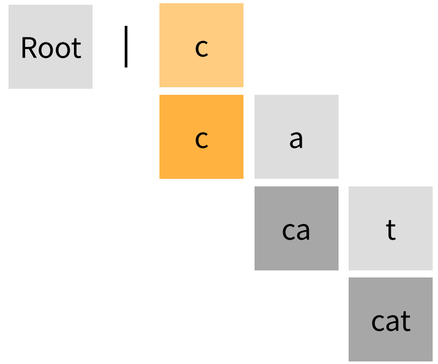




이번엔 문자열 can을 추가한다.

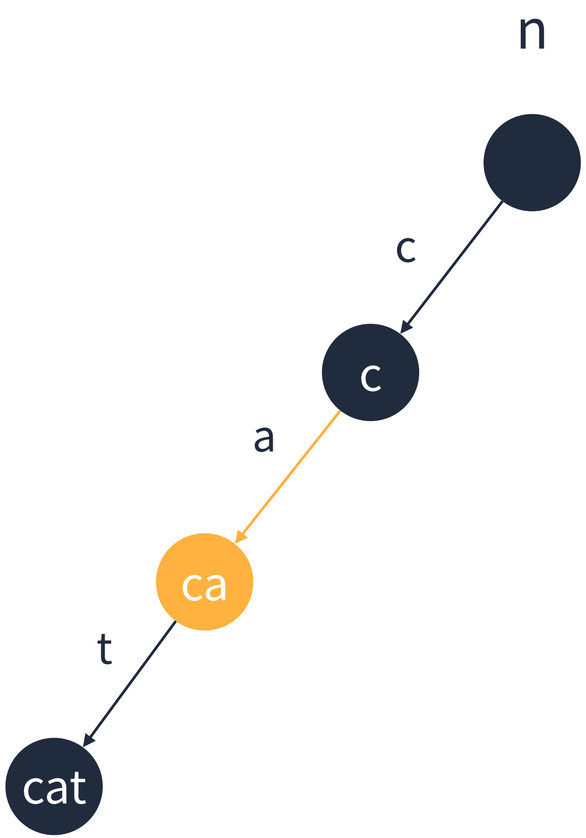
Step 5

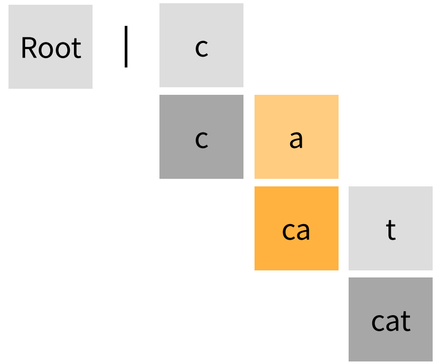




맨 앞 문자열 `c`를 자른 후 확인한다. 이미 정점이 있기에 이동만한다.

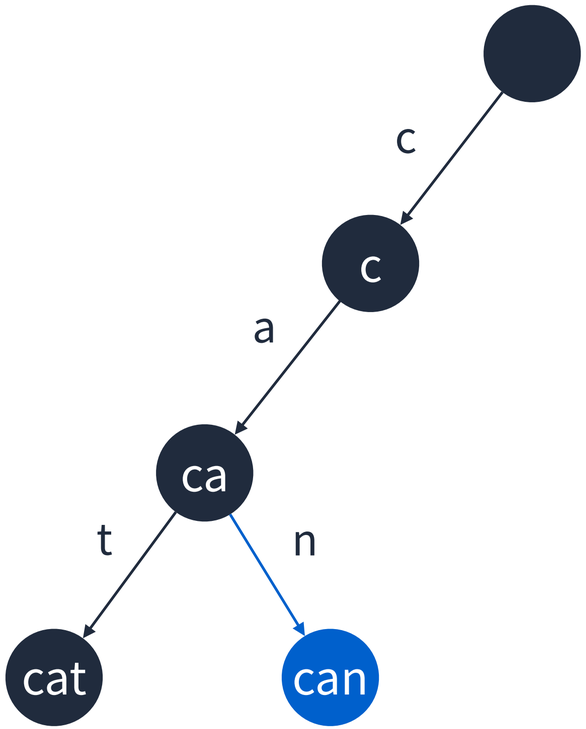
Step 5

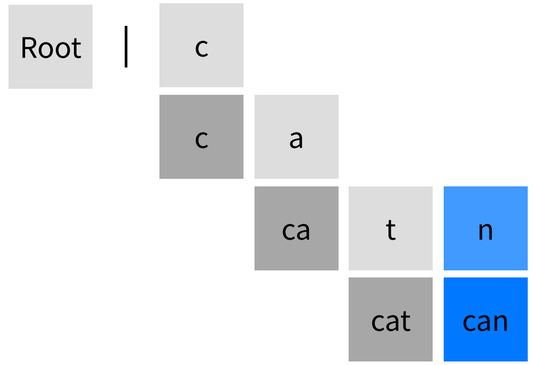




맨 앞 문자열 `a`를 자른 후 확인한다. 이미 정점이 있기에 이동만한다.

Step 5





맨 앞 문자열 `n`를 자른 후 확인한다. n 간선은 없기에 정점을 추가한다.

JavaScript에서 사용법

트라이 구성

```
class Node {
  constructor(value = "") {
    this.value = value;
    this.children = new Map();
  }
}

class Trie {
  constructor() {
    this.root = new Node();
  }

  insert(string) {
    let currentNode = this.root;

    for (const char of string) {
      if (!currentNode.children.has(char)) {
        currentNode.children.set(
          char,
          new Node(currentNode.value + char)
        );
      }

      currentNode = currentNode.children.get(char);
    }
  }

  has(string) {
    let currentNode = this.root;

    for (const char of string) {
      if (!currentNode.children.has(char)) {
        return false;
      }
      currentNode = currentNode.children.get(char);
    }

    return true;
  }
}

const trie = new Trie();
trie.insert("cat");
trie.insert("can");
console.log(trie.has("cat")); // true
console.log(trie.has("can")); // true
console.log(trie.has("cap")); // false
```

트라이 구성

```
class Node {
  constructor(value = "") {
    this.value = value;
    this.children = new Map();
  }
}

class Trie {
  constructor() {
    this.root = new Node();
  }

  insert(string) {
    let currentNode = this.root;

    for (const char of string) {
      if (!currentNode.children.has(char)) {
        currentNode.children.set(
          char,
          new Node(currentNode.value + char)
        );
      }

      currentNode = currentNode.children.get(char);
    }
  }

  has(string) {
    let currentNode = this.root;

    for (const char of string) {
      if (!currentNode.children.has(char)) {
        return false;
      }
      currentNode = currentNode.children.get(char);
    }

    return true;
  }
}

const trie = new Trie();
trie.insert("cat");
trie.insert("can");
console.log(trie.has("cat")); // true
console.log(trie.has("can")); // true
console.log(trie.has("cap")); // false
```

트라이 구성

```
class Node {
  constructor(value = "") {
    this.value = value;
    this.children = new Map();
  }
}

class Trie {
  constructor() {
    this.root = new Node();
  }

  insert(string) {
    let currentNode = this.root;

    for (const char of string) {
      if (!currentNode.children.has(char)) {
        currentNode.children.set(
          char,
          new Node(currentNode.value + char)
        );
      }

      currentNode = currentNode.children.get(char);
    }
  }

  has(string) {
    let currentNode = this.root;

    for (const char of string) {
      if (!currentNode.children.has(char)) {
        return false;
      }
      currentNode = currentNode.children.get(char);
    }

    return true;
  }
}

const trie = new Trie();
trie.insert("cat");
trie.insert("can");
console.log(trie.has("cat")); // true
console.log(trie.has("can")); // true
console.log(trie.has("cap")); // false
```

트라이 구성

```
class Node {
  constructor(value = "") {
    this.value = value;
    this.children = new Map();
  }
}

class Trie {
  constructor() {
    this.root = new Node();
  }

  insert(string) {
    let currentNode = this.root;

    for (const char of string) {
      if (!currentNode.children.has(char)) {
        currentNode.children.set(
          char,
          new Node(currentNode.value + char)
        );
      }

      currentNode = currentNode.children.get(char);
    }
  }

  has(string) {
    let currentNode = this.root;

    for (const char of string) {
      if (!currentNode.children.has(char)) {
        return false;
      }
      currentNode = currentNode.children.get(char);
    }

    return true;
  }
}

const trie = new Trie();
trie.insert("cat");
trie.insert("can");
console.log(trie.has("cat")); // true
console.log(trie.has("can")); // true
console.log(trie.has("cap")); // false
```

트라이 구성

```
class Node {
  constructor(value = "") {
    this.value = value;
    this.children = new Map();
  }
}

class Trie {
  constructor() {
    this.root = new Node();
  }

  insert(string) {
    let currentNode = this.root;

    for (const char of string) {
      if (!currentNode.children.has(char)) {
        currentNode.children.set(
          char,
          new Node(currentNode.value + char)
        );
      }

      currentNode = currentNode.children.get(char);
    }
  }

  has(string) {
    let currentNode = this.root;

    for (const char of string) {
      if (!currentNode.children.has(char)) {
        return false;
      }
      currentNode = currentNode.children.get(char);
    }

    return true;
  }
}

const trie = new Trie();
trie.insert("cat");
trie.insert("can");
console.log(trie.has("cat")); // true
console.log(trie.has("can")); // true
console.log(trie.has("cap")); // false
```

트라이 구성

```
class Node {
  constructor(value = "") {
    this.value = value;
    this.children = new Map();
  }
}

class Trie {
  constructor() {
    this.root = new Node();
  }

  insert(string) {
    let currentNode = this.root;

    for (const char of string) {
      if (!currentNode.children.has(char)) {
        currentNode.children.set(
          char,
          new Node(currentNode.value + char)
        );
      }

      currentNode = currentNode.children.get(char);
    }
  }

  has(string) {
    let currentNode = this.root;

    for (const char of string) {
      if (!currentNode.children.has(char)) {
        return false;
      }
      currentNode = currentNode.children.get(char);
    }

    return true;
  }
}

const trie = new Trie();
trie.insert("cat");
trie.insert("can");
console.log(trie.has("cat")); // true
console.log(trie.has("can")); // true
console.log(trie.has("cap")); // false
```

트라이 구성

```
class Node {
  constructor(value = "") {
    this.value = value;
    this.children = new Map();
  }
}

class Trie {
  constructor() {
    this.root = new Node();
  }

  insert(string) {
    let currentNode = this.root;

    for (const char of string) {
      if (!currentNode.children.has(char)) {
        currentNode.children.set(
          char,
          new Node(currentNode.value + char)
        );
      }

      currentNode = currentNode.children.get(char);
    }
  }

  has(string) {
    let currentNode = this.root;

    for (const char of string) {
      if (!currentNode.children.has(char)) {
        return false;
      }
      currentNode = currentNode.children.get(char);
    }

    return true;
  }
}

const trie = new Trie();
trie.insert("cat");
trie.insert("can");
console.log(trie.has("cat")); // true
console.log(trie.has("can")); // true
console.log(trie.has("cap")); // false
```

programmers

트라이

코딩테스트 광탈방지 A to Z : JavaScript - 이선협 @kciter