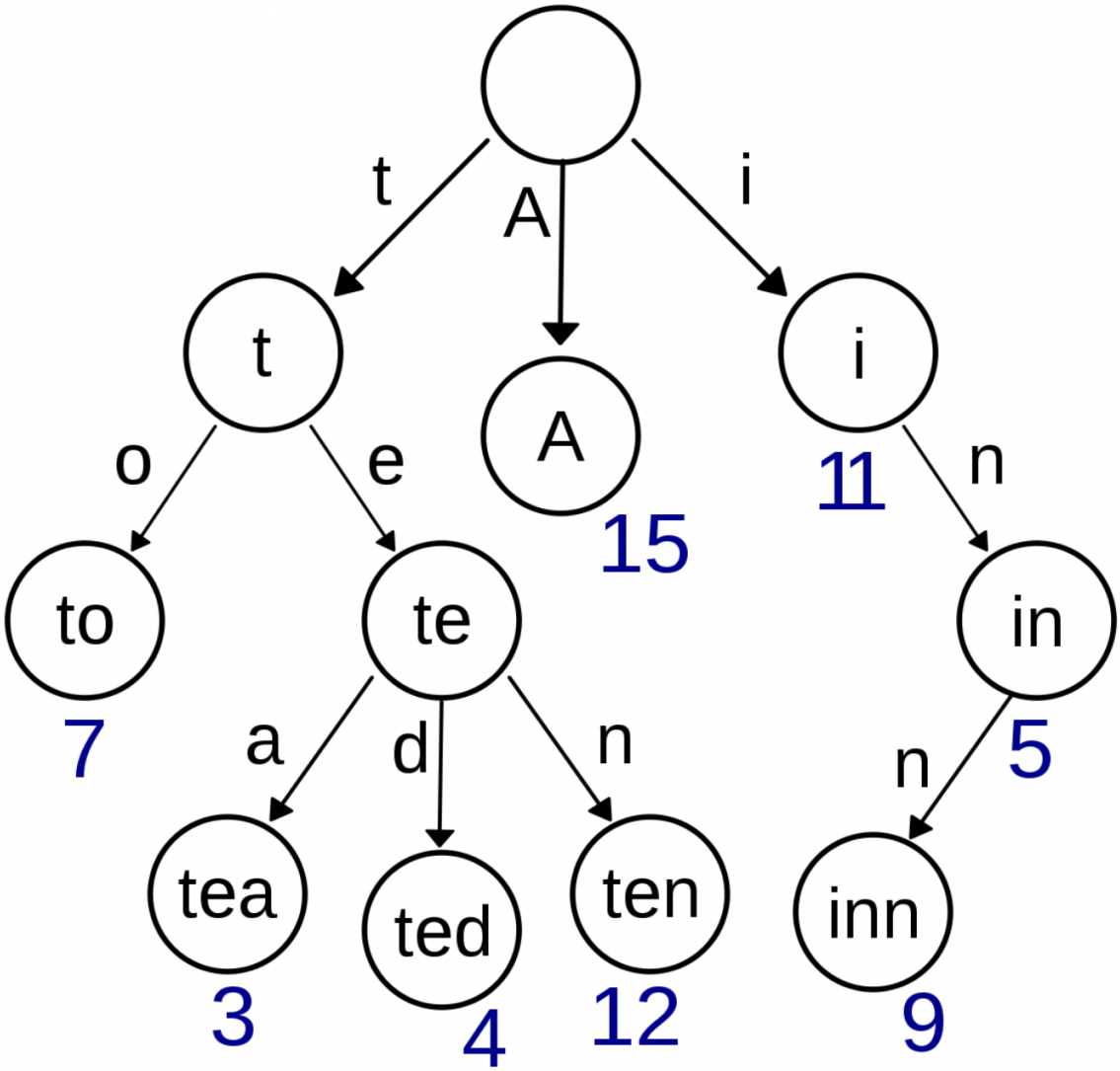
12주차 : Trie

> `트라이(Trie)`는 **문자열을 저장하고 효율적으로 탐색하기 위한** `트리` **형태의 자료구조**이다.
> 래딕스 트리(radix tree) or 접두사 트리(prefix tree) or 탐색 트리(retrieval tree)라고도 한다. `트라이`는 re**trie**val tree에서 나온 단어이다.



목적

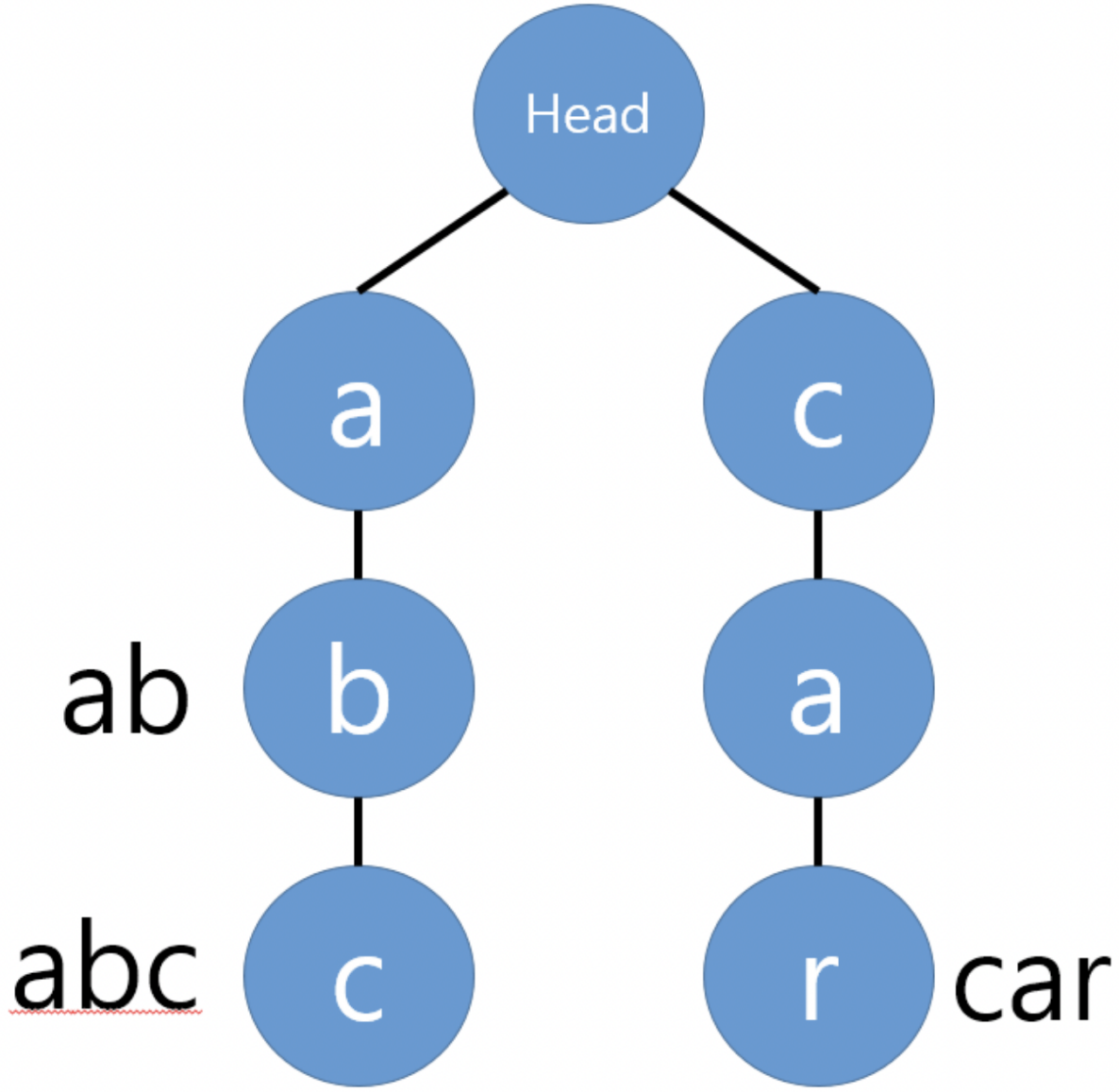
- 사용하는 이유는 문자열의 탐색의 시간복잡도 단축.
- 단, 빠르게 탐색이 가능하다는 장점이 있지만 각 노드에서 자식들에 대한 포인터들을 배열로 모두 저장하고 있다는 점에서 저장 공간의 크기는 크다.
- 검색어 자동완성, 사전에서 찾기 그리고 문자열 검사 같은 부분에서 사용할 수 있다고 첨부된 자료에 나와있습니다.

시간 복잡도

- 제일 긴 문자열의 길이를 `L` 총 문자열의 수를 `M`이라 할 때 시간복잡도는 아래와 같다.
- 생성 시 시간복잡도: `O(M*L)`, 모든 문자열을 넣어야하니 `M`개에 대해서 트라이 자료구조에 넣는건 가장 긴 문자열 길이만큼 걸리니 `L`0만큼 걸려서 `O(M*L)`만큼 걸립니다. 물론 삽입 자체만은 `O(L)`만큼 걸립니다.
- 탐색시 시간복잡도: `O(L)`, worst 의 경우 가장 긴 문자열의 길이 만큼 탐색하기 때문에 `O(L)`만큼 걸린다.

예제를 통한 트라이(Trie)의 이해

'abc', 'ab', 'car' 단어들을 'abc'부터 `트라이`에 저장한다고 가정해보자.



1. **'abc'** `트라이(Trie)`**에 삽입**

Head

Key	None
Data	None
child	{a}

Key	a
Data	None
child	{b}

Key	b
Data	None
child	{c}

Key	c
Data	“abc”
child	{}

- 첫 번째 문자는 'a'이다. 초기에 `트라이` 자료구조 내에는 아무것도 없으므로 Head의 자식노드에 'a'를 추가해준다.
- 'a'노드에도 현재 자식이 하나도 없으므로, 'a'의 자식노드에 'b'를 추가해준다.
- 'c'도 마찬가지로 'b'의 자식노드로 추가해준다.
- 'abc' 단어가 여기서 끝남을 알리기 위해 현재 노드에 abc라고 표시한다. (Data)

2. **'ab'** `트라이(Trie)` **에 삽입**

Head

Key	None
Data	None
child	{a}

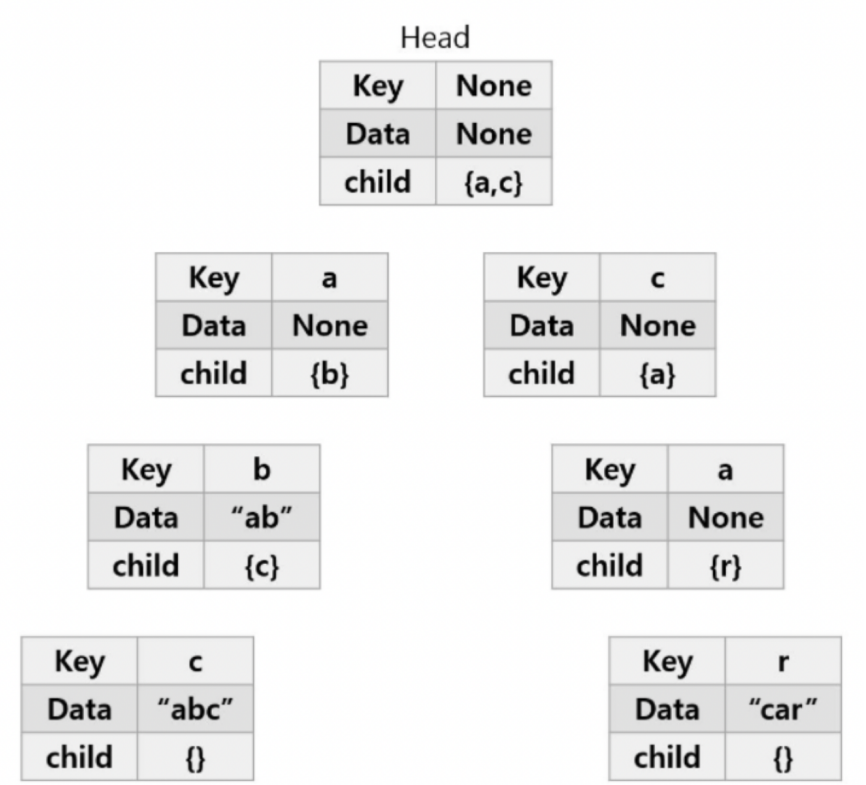
Key	a
Data	None
child	{b}

Key	b
Data	“ab”
child	{c}

Key	c
Data	“abc”
child	{}

- 현재 Head의 자식노드로 'a'가 이미 존재한다. 따라서 'a'노드를 추가하지 않고, 기존에 있는 'a'노드로 이동한다.
- 'b'도 'a'의 자식노드로 이미 존재하므로 'b'노드로 이동한다.
- 'ab' 단어가 여기서 끝이므로 현재 노드에 ab를 표시한다.

3. **'car' `트라이(Trie)`에 삽입**



- Head의 자식노드로 'a'만 존재하고, 'c'는 존재하지 않는다. 따라서 'c'를 자식노드로 추가한다.
- 'c'의 자식노드가 없으므로 마찬가지로 'a'를 추가한다.
- 'a'의 자식노드가 없으므로 마찬가지로 'r'을 추가한다.
- 'car' 단어가 여기서 끝이므로 현재 노드에 car를 표시한다.

참고자료

- https://dev-note-97.tistory.com/171
- https://twpower.github.io/187-trie-concept-and-basic-problem

- https://velog.io/@kimdukbae/자료구조-트라이-Trie