

Sogang

Trie



ICPC Tese

2019-2020 Winter

20141574 임지환 (Sogang University)



- 대표적인 탐색형 자료구조



- 대표적인 탐색형 자료구조

⇒ Binary Search Tree $O(\log n)$



- 대표적인 탐색형 자료구조
⇒ Binary Search Tree $O(\log n)$
- 길이 M 인 문자열의 경우 : $O(M \log N)$



- 대표적인 탐색형 자료구조
⇒ Binary Search Tree $O(\log n)$
- 길이 M 인 문자열의 경우 : $O(M \log N)$



- 메모리 소모up, but $O(M)$ 에 문자열 탐색이 가능한 자료구조



- 최근 기업 문제에서 등장한 Trie문제

2020 kakao blind recruitment P4 ([link](#))

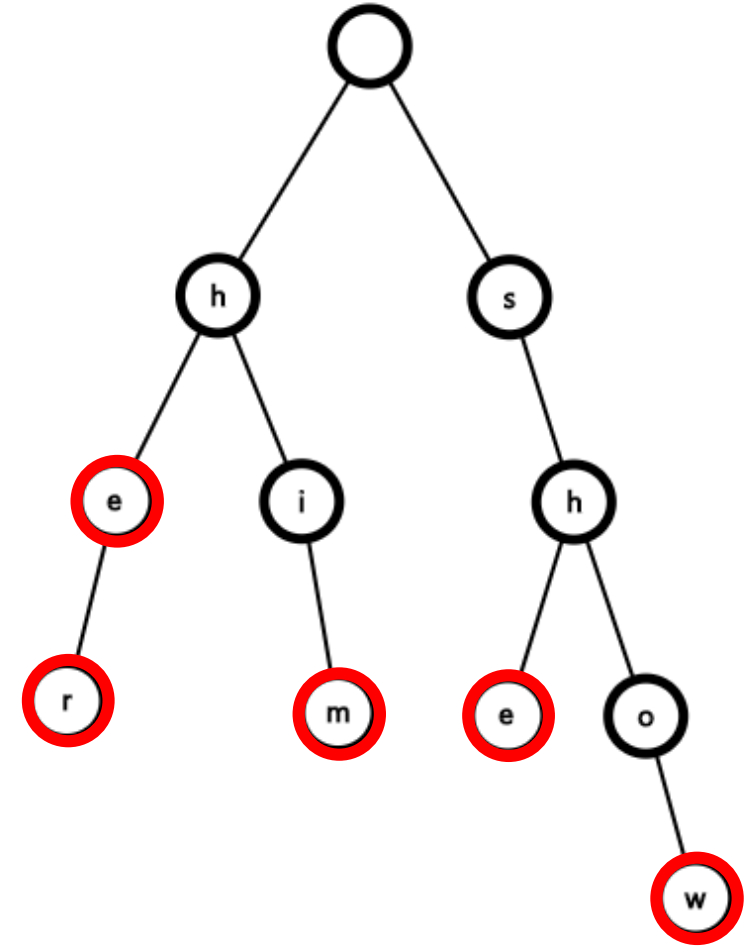
2019 Google CodeJam R.1A P3 ([link](#))

2018 Kickstart R.H P1 ([link](#))

Basic Concept



- 트리 형태
- 각 node마다 한 문자, 다음 문자로 가기 위한 link 포함
- 부모->자식 간 연결을 통해 다음 문자로 이동
- 단어 뿐만 아니라 접두사 또한 저장 가능
- Kind of DFA(Deterministic Finite Automata)



Abstract data type of trie



```
7   const int alphabet = 26;
8   int N, M, cnt;
9   char str[501];
10
11  inline int ctoi(char c) { return c - 'a'; }
12
13  struct Trie {
14      bool terminal;
15      Trie* child[alphabet];
16
17      Trie() : terminal(false) {
18          memset(child, 0, sizeof child);
19      }
20      ~Trie() {
21          for (int i = 0; i < alphabet; i++)
22              delete child[i];
23      }
24      void insert(const char* key) { ... }
32      Trie* find(const char* key) { ... }
38  };
39  Trie* root;
```

20 : bool terminal

- make distinction between a prefix of word and a whole word

21 : Trie* child[26]

- pointers for next node

23 : Trie()

- generator function
- member pointers initialized 0

Trie insertion



```
24 void insert(const char* key) {  
25     if (*key == 0) terminal = true;  
26     else {  
27         int next = ctoi(*key);  
28         if (!child[next]) child[next] = new Trie();  
29         child[next]->insert(key + 1);  
30     }  
31 }
```

24 : terminal = true

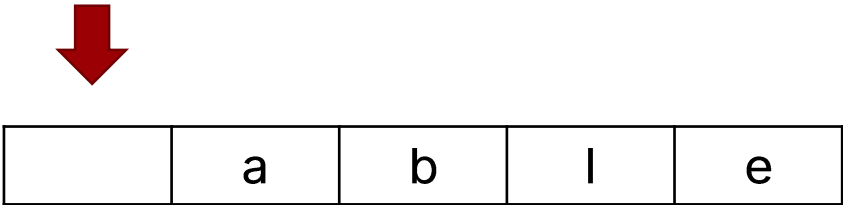
- make distinction between a prefix of word and a whole word

27 : next = ctoi(*key)

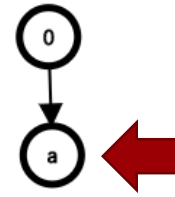
28 : child[next] == NULL

- No route to go

Trie insertion

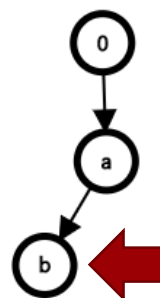


Trie insertion



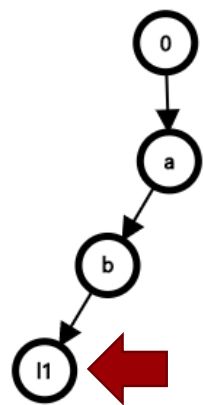
	a	b	l	e
--	---	---	---	---

Trie insertion



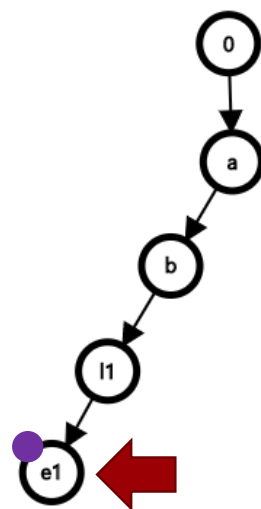
	a	b	l	e
--	---	---	---	---

Trie insertion




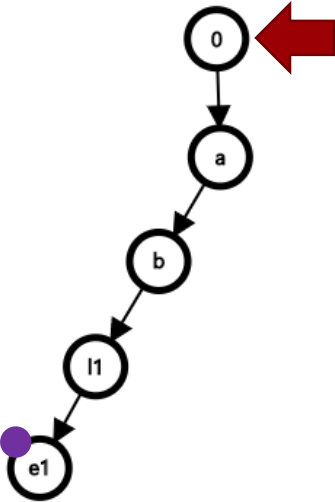
	a	b	l	e
--	---	---	---	---

Trie insertion



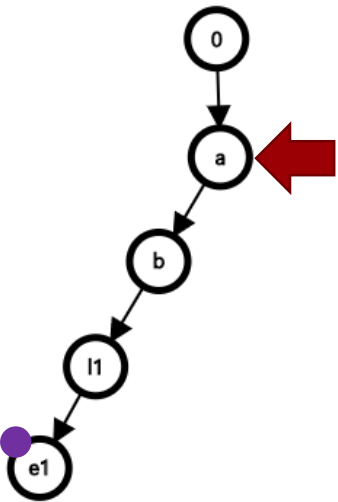
	a	b	l	e
--	---	---	---	---

Trie insertion



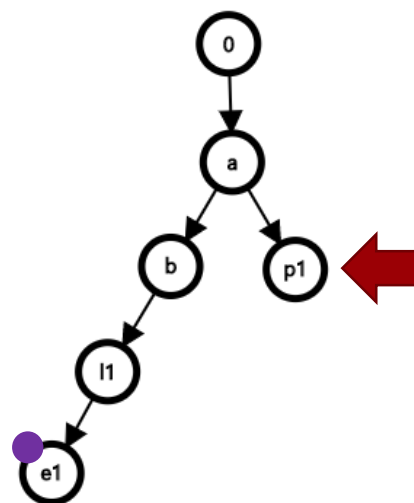
	a	p	p	l	y
--	---	---	---	---	---

Trie insertion



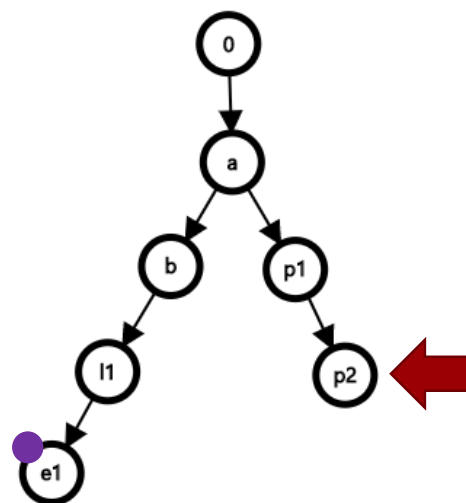
	a	p	p	l	y
--	---	---	---	---	---

Trie insertion



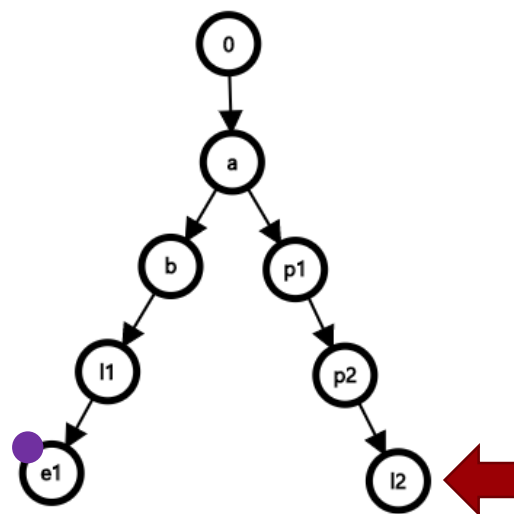
	a	p	p	l	y
--	---	---	---	---	---

Trie insertion



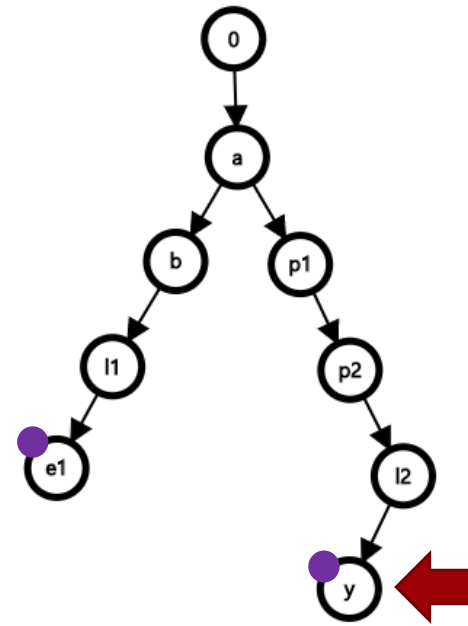
	a	p	p	l	y
--	---	---	---	---	---

Trie insertion



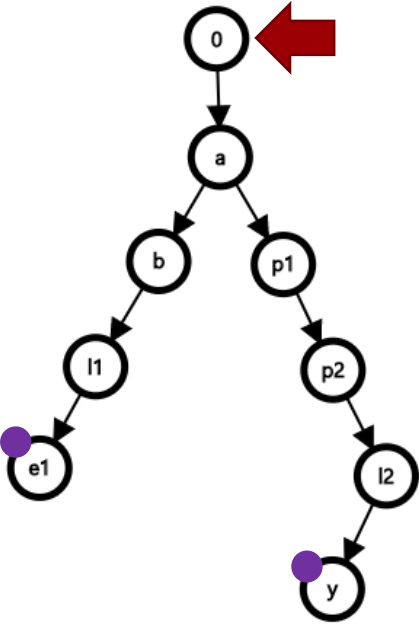
	a	p	p	l	y
--	---	---	---	---	---

Trie insertion



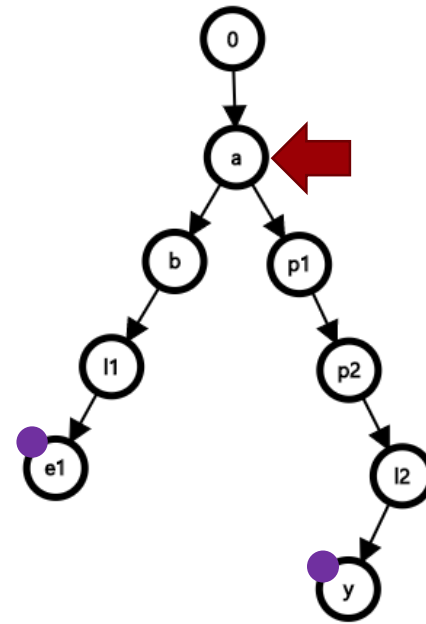
	a	p	p	l	y
--	---	---	---	---	---

Trie insertion



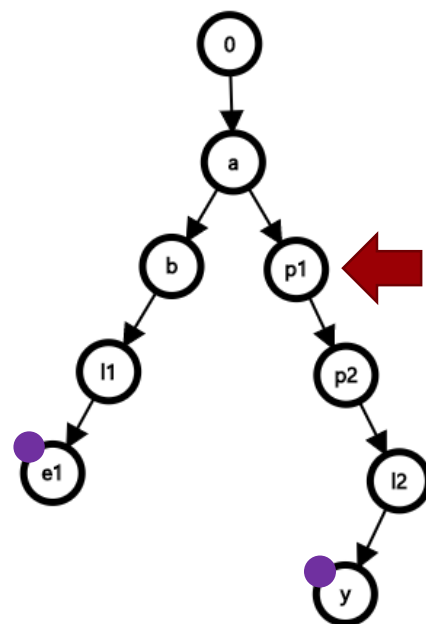
	a	p	p	l	e
--	---	---	---	---	---

Trie insertion



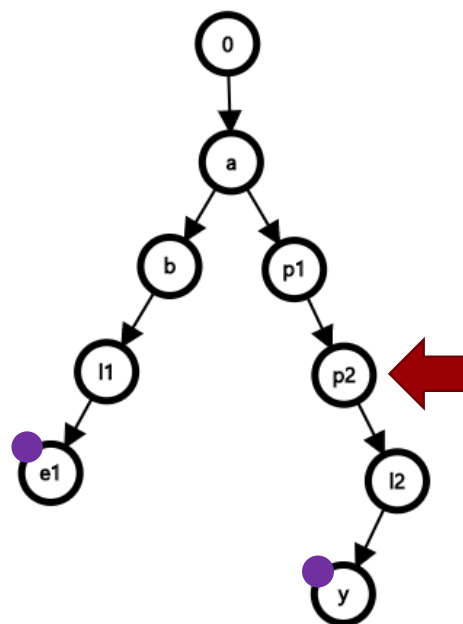
	a	p	p	l	e
--	---	---	---	---	---

Trie insertion



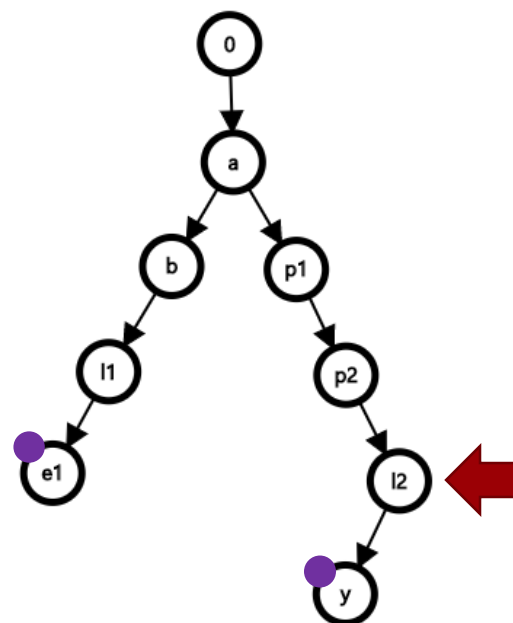
	a	p	p	l	e
--	---	---	---	---	---

Trie insertion



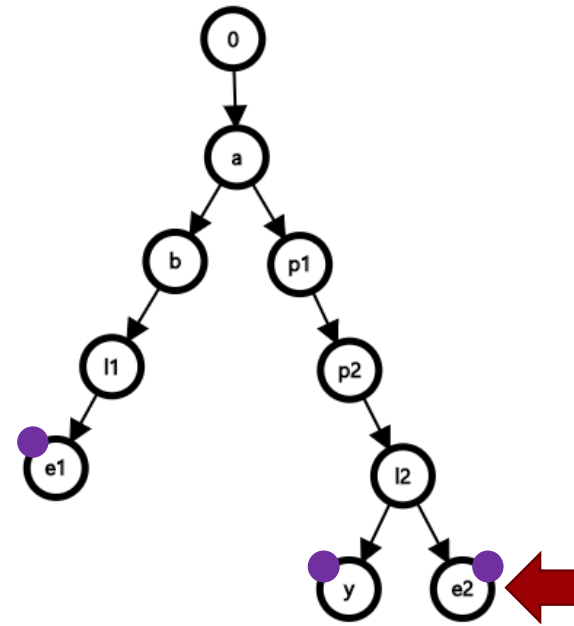
	a	p	p	l	e
--	---	---	---	---	---

Trie insertion



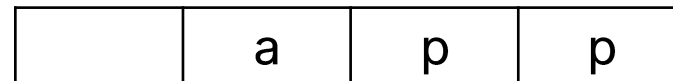
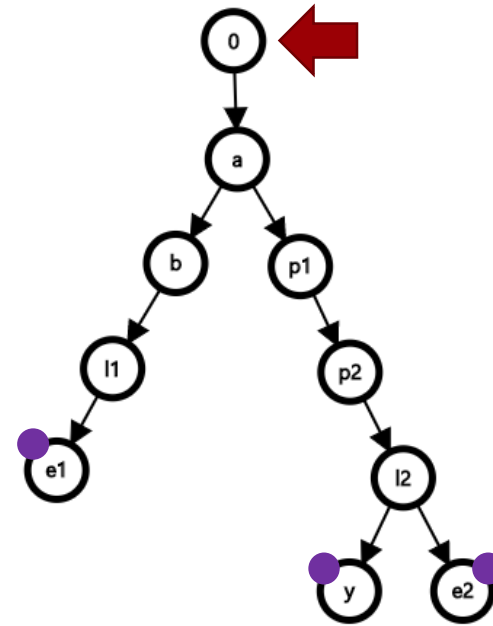
	a	p	p	l	e
--	---	---	---	---	---

Trie insertion

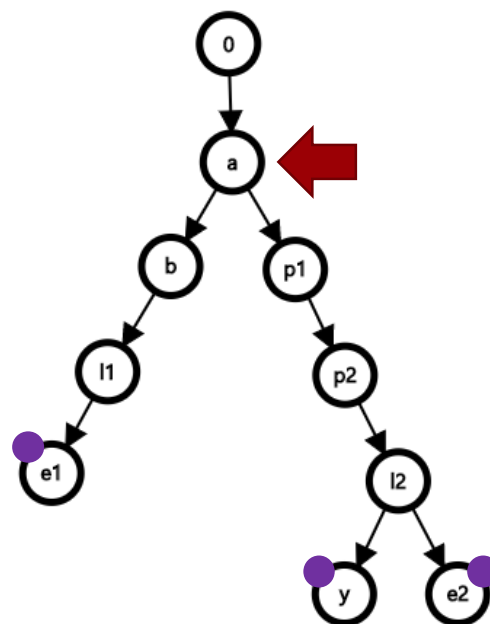


	a	p	p	l	e
--	---	---	---	---	---

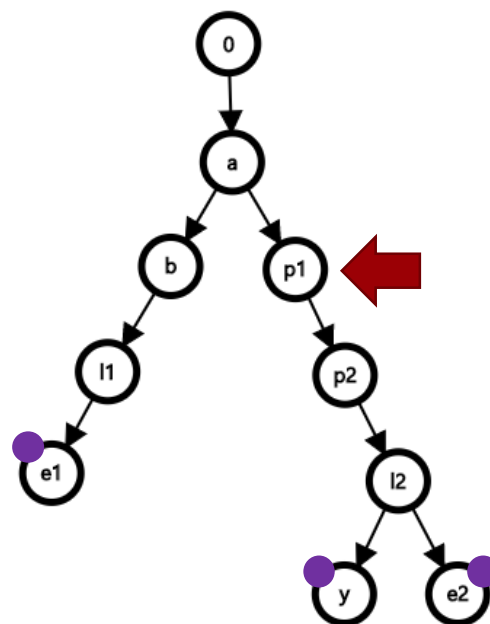
Trie insertion



Trie insertion

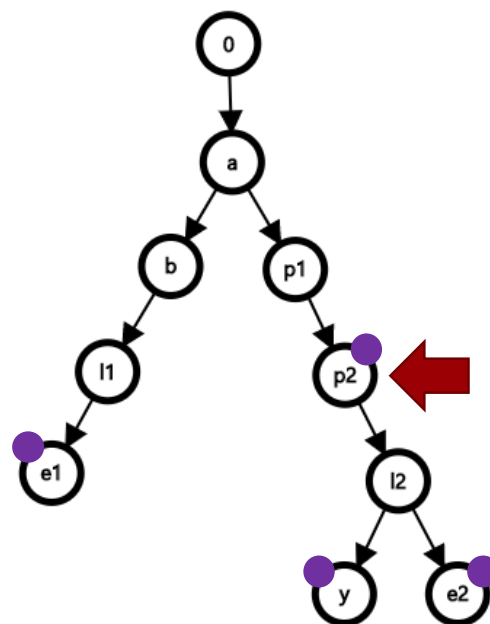


Trie insertion



	a	p	p
--	---	---	---

Trie insertion



	a	p	p
--	---	---	---

search



```
32 Trie* find(const char* key) {  
33     if (*key == 0) return this;  
34     int next = ctoi(*key);  
35     if (!child[next]) return NULL;  
36     return child[next]->find(key + 1);  
37 }  
38 };  
39 Trie* root;
```

```
41 int solve() {  
42     int ret = M;  
43     for (int i = 0; i < M; i++) {  
44         scanf("%s", str);  
45         Trie* isthere = root->find(str);  
46         if (isthere == NULL || isthere->terminal == false) ret--;  
47     }  
48  
49     return ret;  
50 }
```

33 : End of string

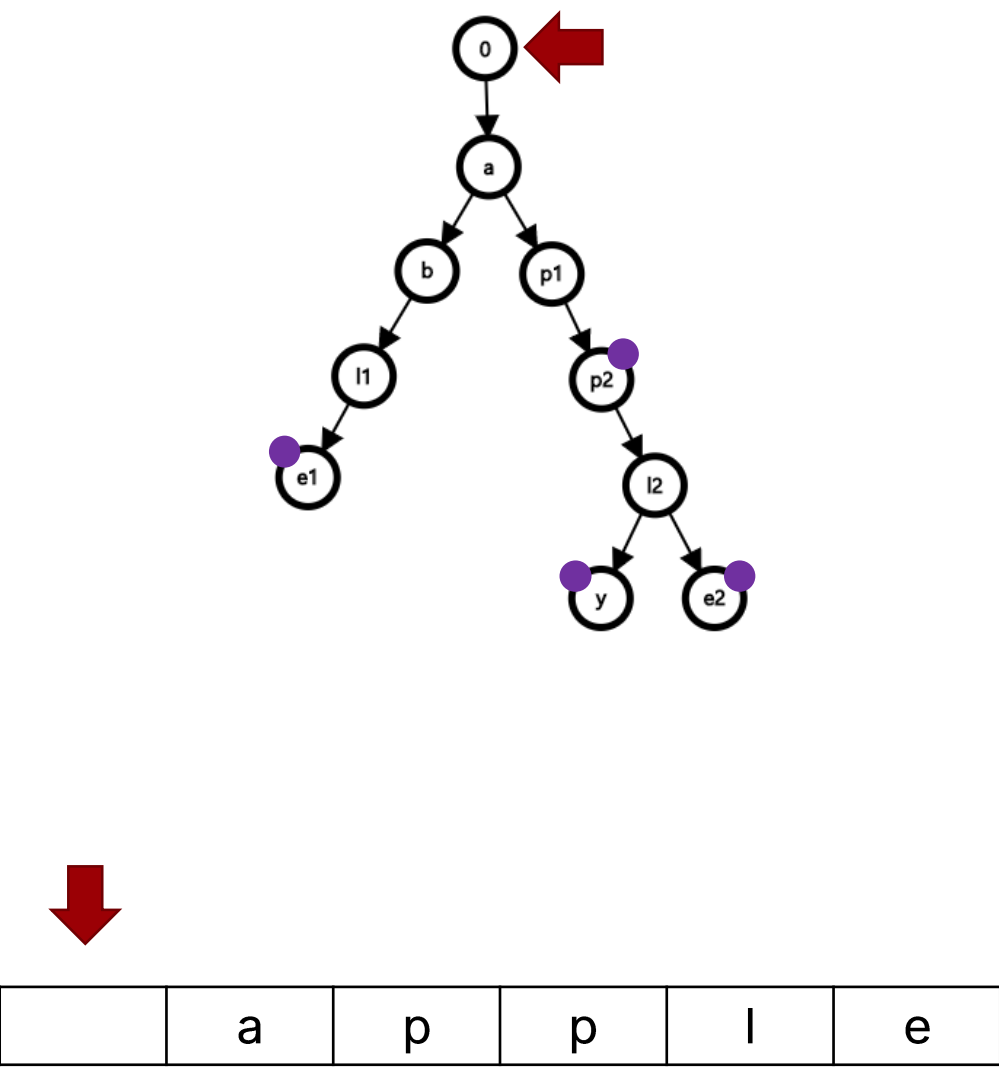
- Line 46 if (isthere->terminal == false)

35 : No ways to find

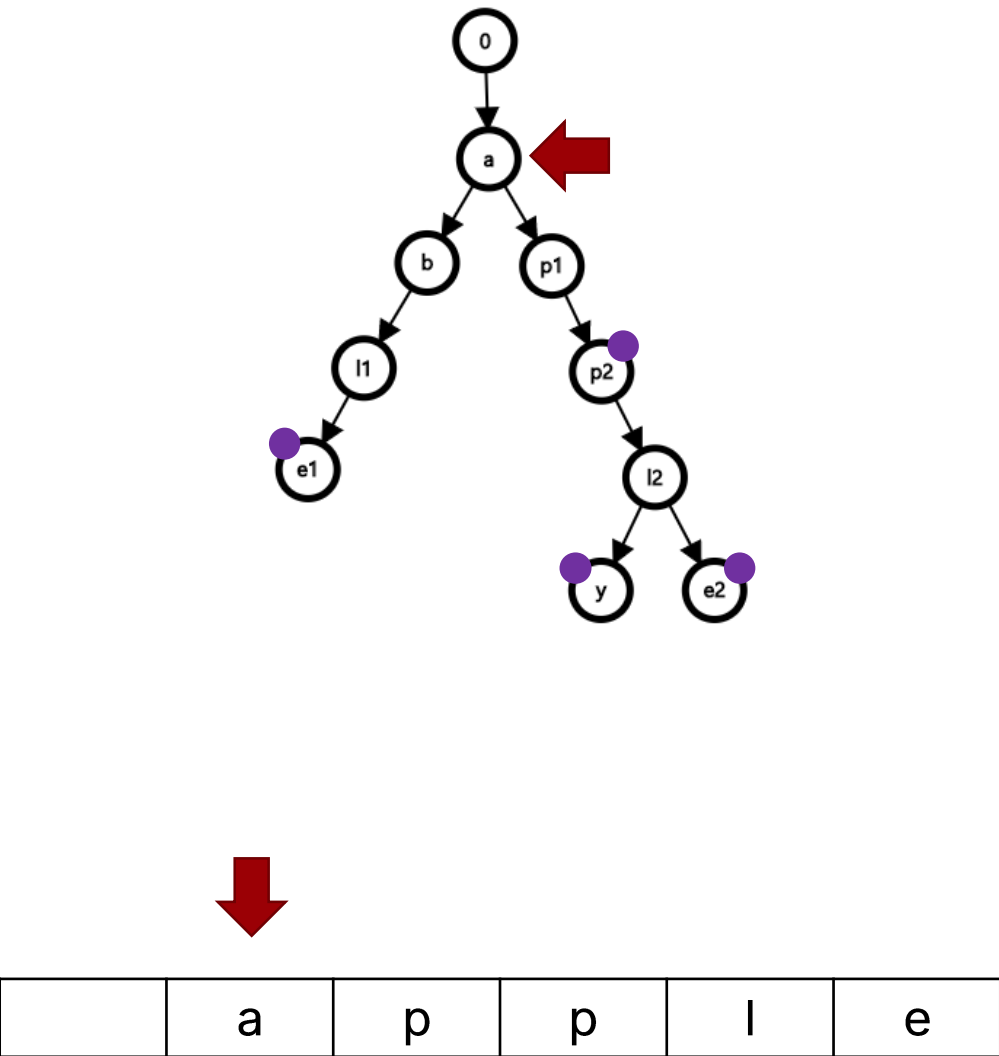
- Line 46 if (isthere == NULL)

36 : can find ways to go

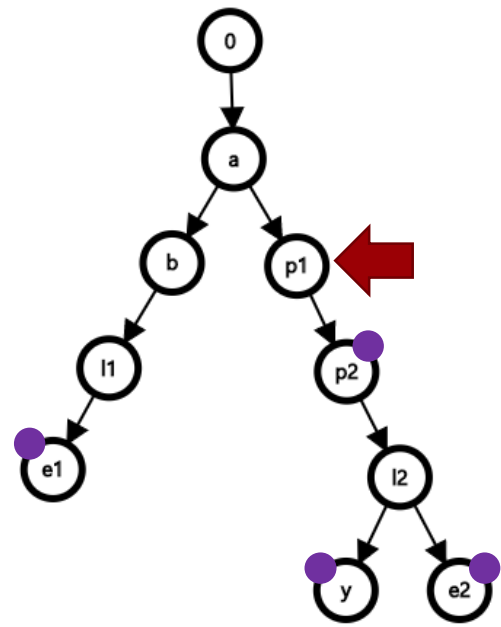
search



search

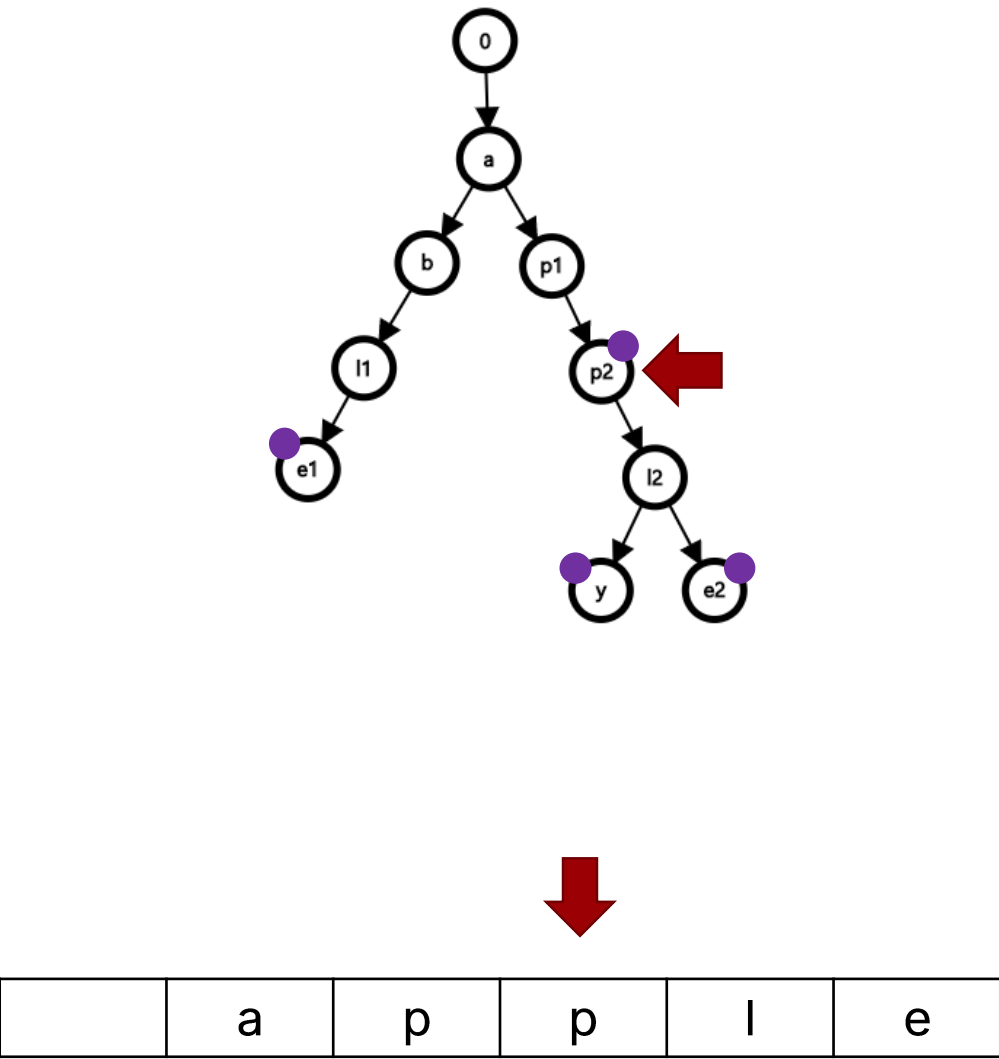


search

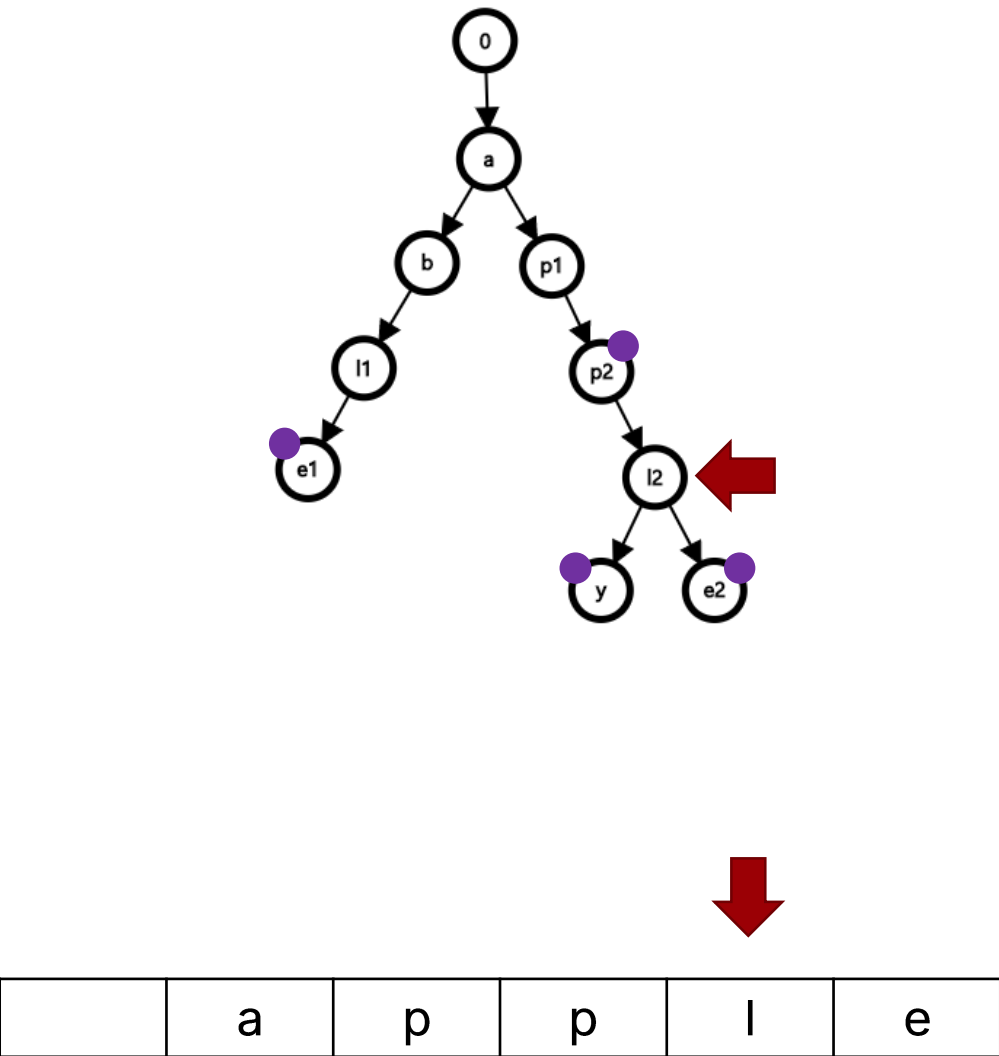


	a	p	p	l	e
--	---	---	---	---	---

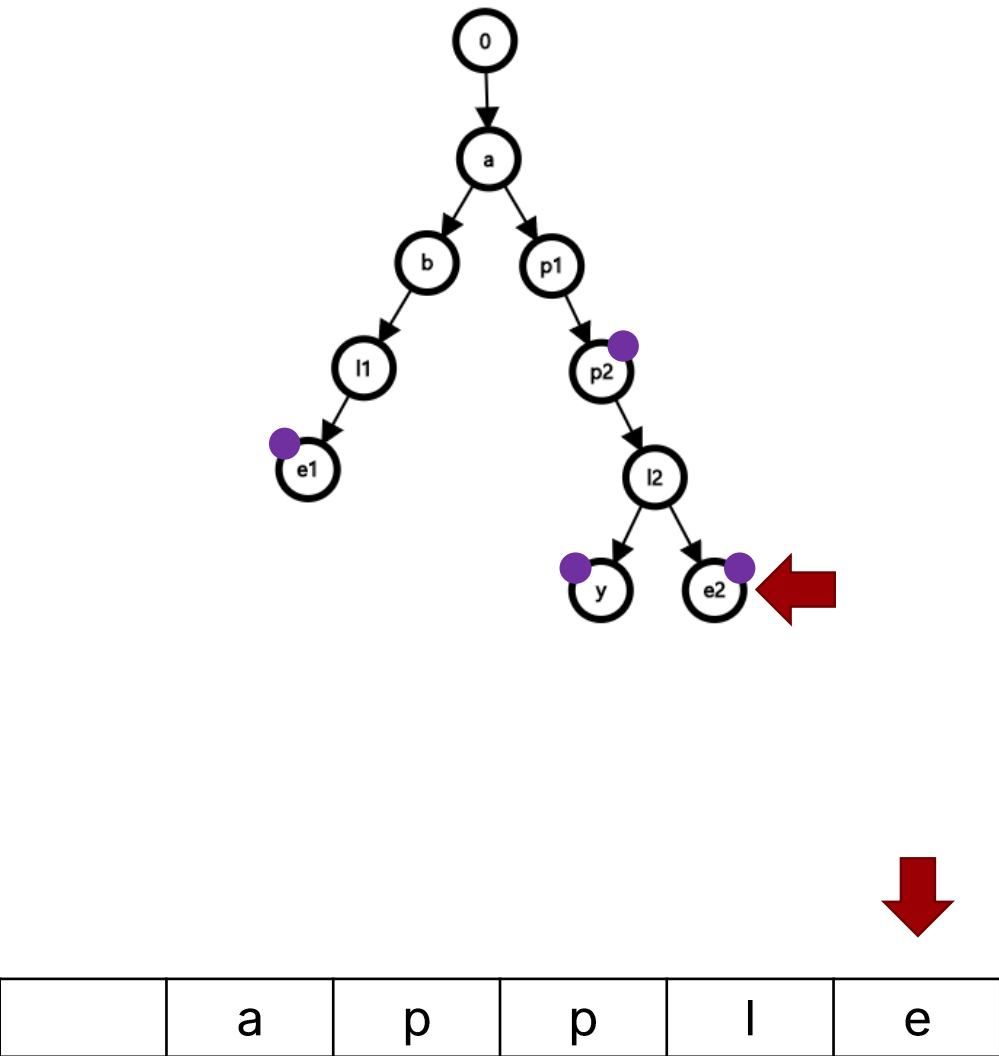
search



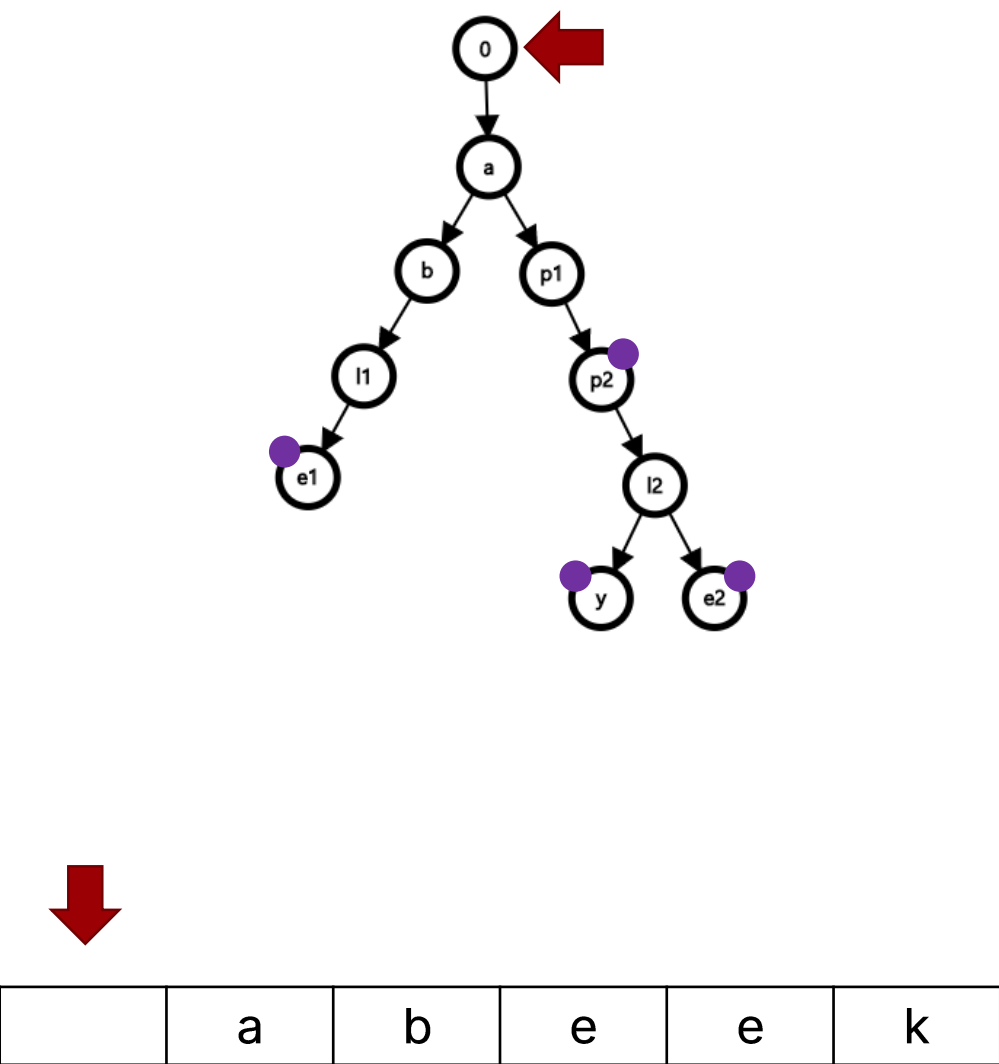
search



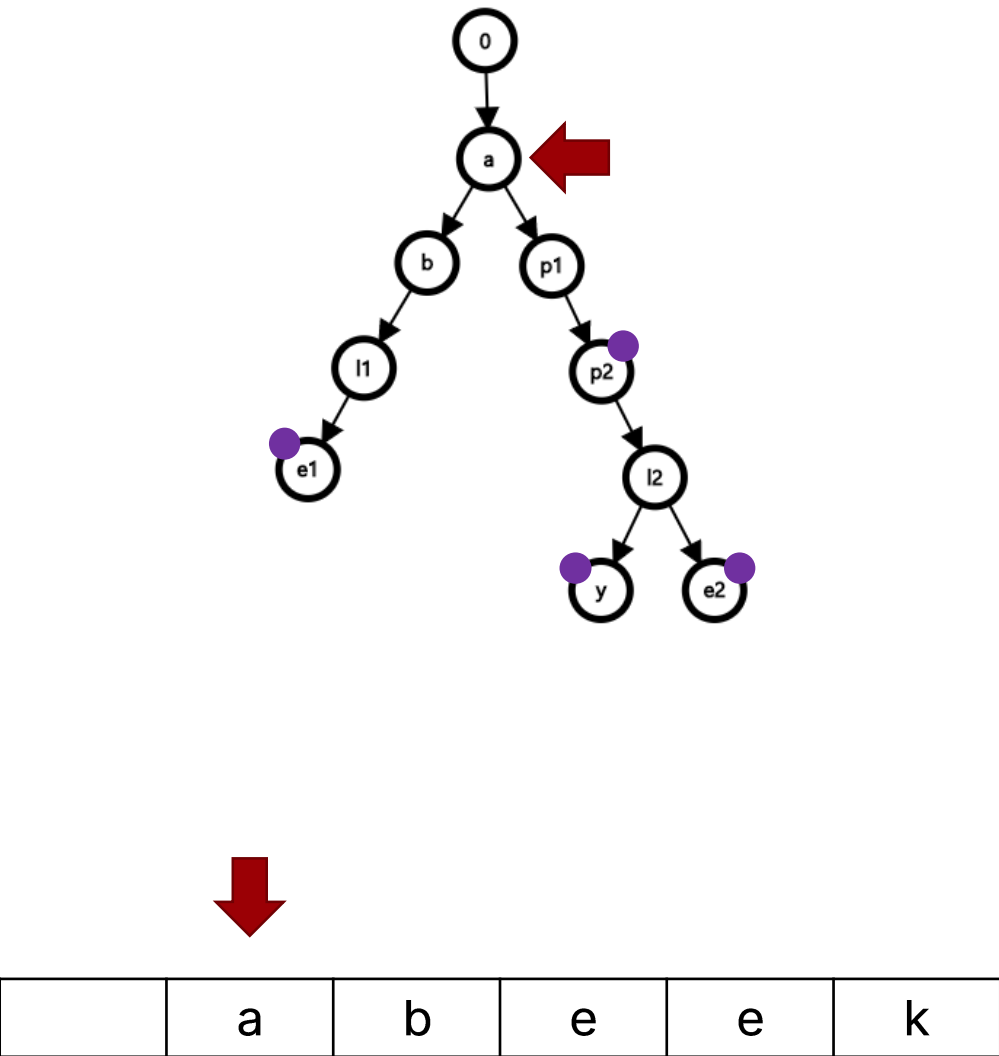
search



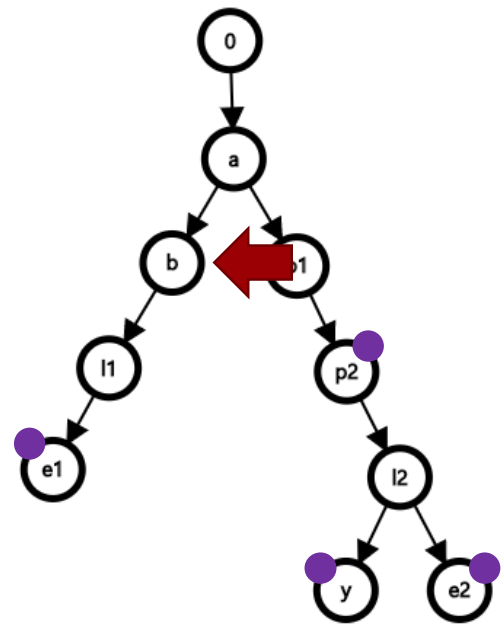
search



search

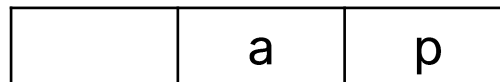
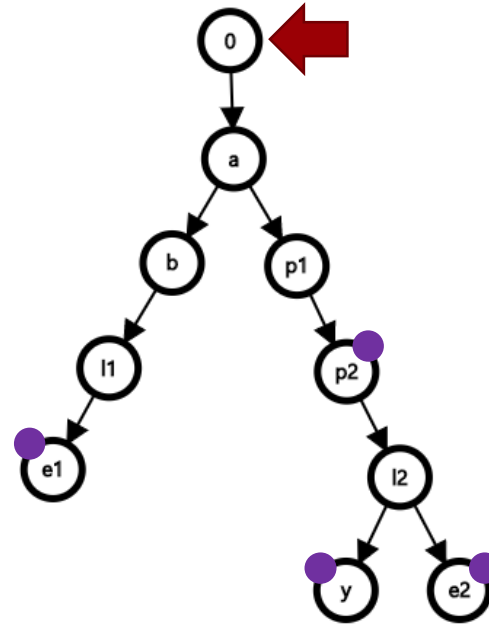


search

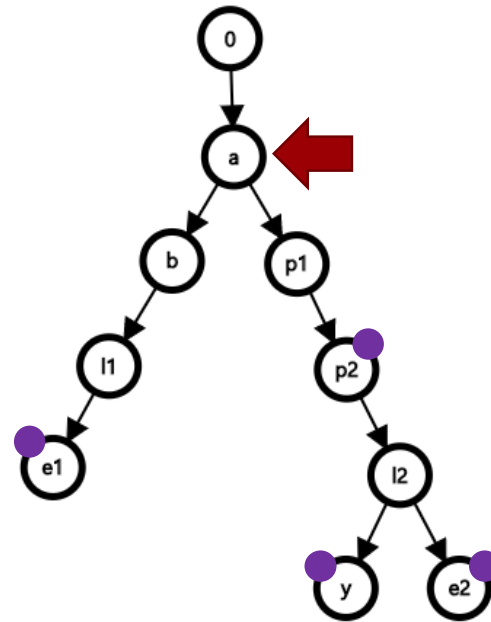


	a	b	e	e	k
--	---	---	---	---	---

search

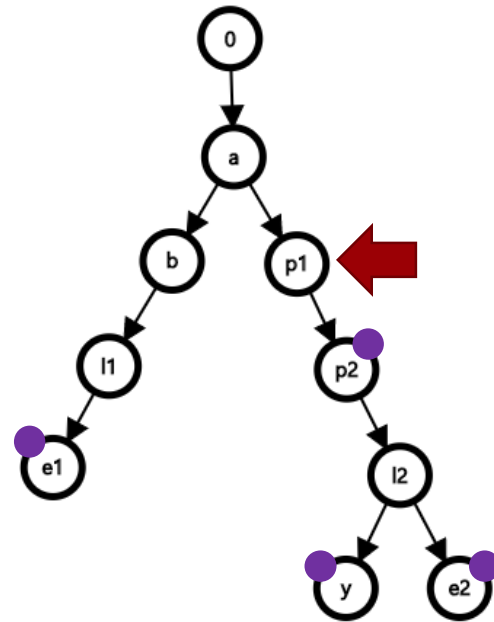


search



	a	p
--	---	---

search



	a	p
--	---	---

Implementation by static array



	Root	a	b	l	e	p	p	l	y	e
0=a	1									
1=b		2								
2=c										
3=d										
4=e				4				9		
...										
11=l			3				7			
...										
15=p		5				6				
...										
24=y								8		
25										

termin

				v		v		v	v
--	--	--	--	---	--	---	--	---	---

#9202 Boggle



- 4×4 board에서 단어장에 존재하는 단어 찾기 게임
- 단어의 수 w ($1 < w < 300,000$), 길이 l ($0 < l < 8$)
- 보드의 개수 b ($1 < b < 30$), 대문자로만 구성
- board 내 이동은 인접한 가로, 세로, 대각선으로만 가능 (8방향)
- 단어 길이당 배점
 - 1, 2 : 0점
 - 3, 4 : 1점
 - 5 : 2점
 - 6 : 3점
 - 7 : 5점
 - 8 : 11점
- 얻을 수 있는 최대 점수, 가장 긴 단어, 찾은 단어의 수를 구해보자.

Word list

- ICPC
- ACM
- CONTEST
- GCPC
- PROGRAM

A	C	M	A
A	P	C	A
T	O	G	I
N	E	S	T



- 모든 경우의 수는 몇 개일까?



- 모든 경우의 수는 몇 개일까?
 1. 보드의 수 : 30개



- 모든 경우의 수는 몇 개일까?
 1. 보드의 수 : 30개
 2. 시작 지점 : 16개



- 모든 경우의 수는 몇 개일까?
 1. 보드의 수 : 30개
 2. 시작 지점 : 16개
 3. 한 점에서 나올 수 있는 단어의 개수 : $< 7^7$ (= 823,543)



- 모든 경우의 수는 몇 개일까?

1. 보드의 수 : 30개
2. 시작 지점 : 16개
3. 한 점에서 나올 수 있는 단어의 개수 : $< 7^7 (= 823,543)$

$$\prod = 395,300,640$$



- 모든 경우의 수는 몇 개일까?

1. 보드의 수 : 30개
2. 시작 지점 : 16개
3. 한 점에서 나올 수 있는 단어의 개수 : $< 7^7 (= 823,543)$

$$\prod = 395,300,640$$

- 단어 길이의 기댓값 : 4.5, trie로 위의 모든 경우에 대해 탐색을 시도한다면?

$$1,778,852,880 (< 1.8 \times 10^9)$$



- 모든 경우의 수는 몇 개일까?

1. 보드의 수 : 30개
2. 시작 지점 : 16개
3. 한 점에서 나올 수 있는 단어의 개수 : $< 7^7 (= 823,543)$

$$\prod = 395,300,640$$

- 단어 길이의 기댓값 : 4.5, trie로 위의 모든 경우에 대해 탐색을 시도한다면?

$$1,778,852,880 (< 1.8 \times 10^9)$$

- 시간제한 : 10초



- 문제 해결 과정

- Trie로 해결 가능함을 알았을 경우

1. Trie 생성, 단어 리스트 추가



- 문제 해결 과정

- Trie로 해결 가능함을 알았을 경우

1. Trie 생성, 단어 리스트 추가
2. 구해야 하는 정보 확인(점수, 가장 긴 단어, 찾은 단어의 수)



- 문제 해결 과정

- Trie로 해결 가능함을 알았을 경우

1. Trie 생성, 단어 리스트 추가
2. 구해야 하는 정보 확인(점수, 가장 긴 단어, 찾은 단어의 수)
3. 모든 점에 대하여, 한점으로부터 시작하여 나올 수 있는 모든 케이스 추출



- 문제 해결 과정

- Trie로 해결 가능함을 알았을 경우

1. Trie 생성, 단어 리스트 추가
2. 구해야 하는 정보 확인(점수, 가장 긴 단어, 찾은 단어의 수)
3. 모든 점에 대하여, 한점으로부터 시작하여 나올 수 있는 모든 케이스 추출(Backtracking)



- 문제 해결 과정

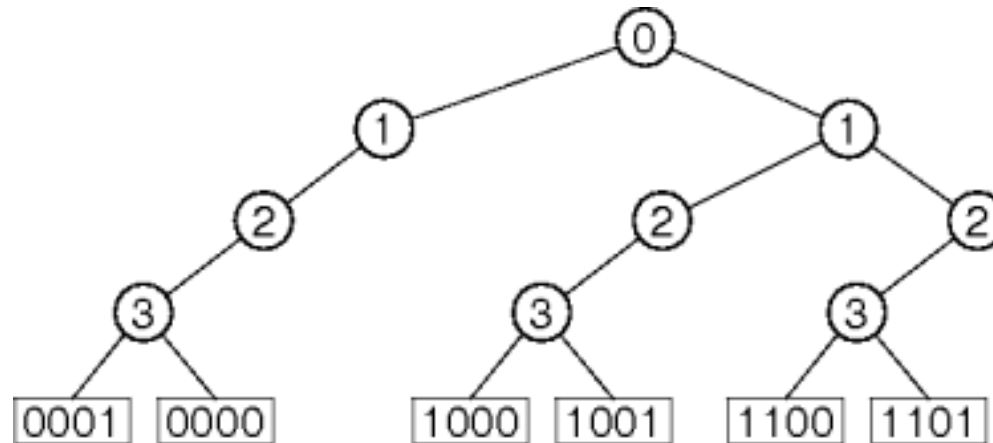
- Trie로 해결 가능함을 알았을 경우

1. Trie 생성, 단어 리스트 추가
2. 구해야 하는 정보 확인(점수, 가장 긴 단어, 찾은 단어의 수)
3. 모든 점에 대하여, 한점으로부터 시작하여 나올 수 있는 모든 케이스 추출(Backtracking)
4. 중복하여 찾은 경우 방지를 위한 설계 필요

Binary trie



- Branch가 0 또는 1만 있는 trie
- 수의 탐색을 상수시간에 가능



#13505 두 수 XOR



- N ($2 \leq N \leq 100,000$)개의 수 A_1, A_2, \dots, A_N ($0 \leq A_i \leq 10^9$)가 주어졌을 때
- XOR한 값이 가장 큰 두 수를 찾아라.

#13505 두 수 XOR



- 두 수의 xor연산

1	0	1	1	0	0	1	1	0	1
0	1	1	1	1	0	0	1	1	0

#13505 두 수 XOR



- 두 수의 xor연산

1	0	1	1	0	0	1	1	0	1
0	1	1	1	1	0	0	1	1	0
<hr/>									
1	1	0	0	1	0	1	0	1	1

#13505 두 수 XOR



- 두 수의 xor연산

1	0	1	1	0	0	1	1	0	1
0	1	1	1	1	0	0	1	1	0
<hr/>									
1	1	0	0	1	0	1	0	1	1

- 하위 비트들의 합 < 상위 비트 1개의 값



- 문제 해결 과정

1. 주어진 수들을 binary trie에 추가



- 문제 해결 과정

1. 주어진 수들을 binary trie에 추가
2. 각각의 수들에 대하여, 상위 비트가 최대한 다르도록 경로 추적(each $O(1)$)
 - 1) 현재 비트와 다른 비트로 가는 경로가 있는 경우 해당 크기만큼 +
 - 2) 경로가 없는 경우 값 추가 없이 탐색



- 문제 해결 과정

1. 주어진 수들을 binary trie에 추가
2. 각각의 수들에 대하여, 상위 비트가 최대한 다르도록 경로 추적(each $O(1)$)
 - 1) 현재 비트와 다른 비트로 가는 경로가 있는 경우 해당 크기만큼 +
 - 2) 경로가 없는 경우 값 추가 없이 탐색
3. 구한 각 값들 중 최대값 추출



#14425 문자열 집합

#4358 생태학

#5052 전화번호 목록

#9202 Boggle

#3172 현주와 윤주의 재미있는 단어 게임

#5446 용량 부족

#13505 두 수 XOR

#16902 mex

#16903 수열과 쿼리 20