[22.01.14] 워드 프로세서

TS워드

김 태 현

문 제

void init(int N, char str[])

길이 N(<=40,000)의 str이 주어짐
'_' 기준으로 3~10자리 소문자 단어 존재

void addDict(char word[]) :1,000회

3~10자리의 소문자 단어가 사전에 등록

void removeDict(char word[]) : 500회

사전에서 word[] 삭제

int correctWord(int start, int end) : 500회

str의 [start, end] 구간 단어들을 아래 기준에 맞게 교정
- 사전에 등록되어 있지 않는 경우
- 사전의 단어 중 문자 한 개만 다른 단어로 교정
- 여러 개면 사전순 빠른 단어

return : 교정한 단어의 수

void destroy(char result[])

str의 최종 상태 출력

- str의 단어 개수? 40,000 / 4 = 10,000개
- 사전의 단어 개수? 1,000개
- [start, end]는 전체 구간 일 수 있음

  = 매번 10,000 * 1,000 번의 단어 비교를 해야 할 수 있음

Naive

- 사전을 그냥 배열로 등록하고 매번 [start, end] 구간의 단어들과 일일이 비교한다.
- 같은 단어 파악, 문자 한 개 다른 단어 중 사전순으로 가장 빠른 단어 파악
- O(str[start:end] 단어 수 * 사전 단어 수 * 문자열 길이) * correctWord() 호출 횟수

  = worst case O(10,000 * 1,000 * 10) * 500

  average case O(5,000 * 1,000 * 5) * 500

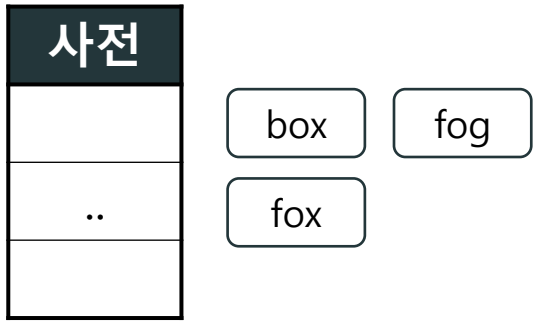
str

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
c	a	p	_	b	o	b	_	f	o	z	_	b	o	b	o	b

사전
box
fox
fog

사전에 단어가 존재하는지 찾는 방법?

- 전체 탐색은 오래 걸리므로 hash!
- 사전을 hash로 등록하고 hash 검색



str

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
c	a	p	_	b	o	b	_	f	o	z	_	b	o	b	o	b

사전에 한 자 다른 단어가 존재하는지 찾는 방법?

- **등록되어 있는 사전을 활용한다면**

  단어의 각 자리를 다른 문자로 교체 후 hash 검색

  : O(단어 수 * 단어 길이 * 소문자 수 * hash 검색 비용) * 호출 횟수

  = O(10,000 * 10 * 26 * (10+a)) * 500

  느림

- **검색을 유리하게 자료를 저장해보자**

  사전 단어 등록 시, 한 자리가 달라도 검색이 가능하도록

  1. 각 자리를 모든 문자로 변경하여 분류/등록한다면?
  2. 각 자리를 조커 문자로 변경하여 분류/등록한다면?

사전에 한 자 다른 단어가 존재하는지 찾는 방법?

1. 각 자리를 모든 문자로 변경하여 분류/등록한다면?

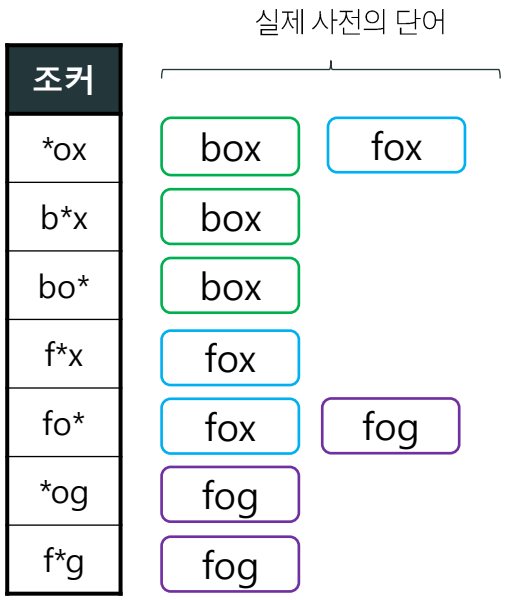
사전의 단어마다 (단어 길이 * 소문자 수) = (10 * 26) 개의 노드가 등록/삭제되어야 한다.
총 등록되는 노드 수 = 1,000 * 10 * 26 = 260,000개
ex) box => aox ~ zox , bax ~ bzx , boa ~ boz

검색 시, 단어에 등록된 사전의 단어들만 확인한다.

2. 각 자리를 조커 문자로 변경하여 분류/등록한다면?

사전의 단어마다 (문자열 길이) = (10) 개의 노드가 등록/삭제되어야 한다.
총 등록되는 노드 수 = 1,000 * 10 = 10,000개
ex) box => *ox , b*x , bo*

검색 시, 단어의 각 자리를 조커 문자로 변경한 단어에 등록된 사전의 단어들만 확인한다.



검색량은 1번이 조금 적을 수 있겠으나 등록되는 노드 수가 훨씬 많아 연산비용이 증가할 수 있다.

str 관리

1,2 상관 없음

1. 주어진 대로 관리

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
c	a	p	_	b	o	b	_	f	o	z	_	b	o	b	o	b	_

초기에 str의 마지막에 _를 추가
start ~ end+1 까지 돌며 단어 생성, _나오면 단어 처리 후 단어 초기화 하며 진행

2. 단어별로 분류

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
str	c	a	p	_	b	o	b	_	f	o	z	_	b	o	b	o	b	_	
id	0				1				2				3						4

str : start ~ end => **word** : id[start] ~ id[end+2] - 1

id	word
0	cap
1	bob
2	foz
3	bobbob

필요한 데이타셋

unordered_set<string> dict

사전
..

box fog
fox

unordered_map<string, int> dict2

조커	id
*ox	0
b*x	1
bo*	2
f*x	3
fo*	4
*og	5
f*g	6

vector<string> v[1,000 * 10]
set<string> s[1,000 * 10]

id
0
1
2
3
4
5
6

box fox
box
box
fox
fox fog
fog
fog

str

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
c	a	p	_	b	o	b	_	f	o	z	_	b	o	b	o	b	_

vector? set?
set을 이용하면 필요한 id 별로 가장 우선순위 높은 단어만 비교하면 된다.
but, id별 노드 수는 대부분 그리 크지 않을 것으로 예상된다.
따라서, set의 이점이 발휘되기보다는 본연의 느림으로 비효율이지 않을까?
input data에 따라 달라질듯?

속도 개선

1. **non-stl 버전으로 작성**
2. **string을 최대한 배제**
    - 문자열 저장은 char* 를 이용
    - key값을 string -> unsigned long long
    - ‘a’=1 , ‘z’=26 , ‘*’=27 로 표현하여 28진법 활용
    - 0 ~ 28^{10}-1 범위로 나타내지므로 unsigned long long 범위에 들어온다.


감사합니다