숙제 2 : 나만의 dictionary 만들기

<due : 05.09 23:59>

이번 숙제에서는 26-ary (ordered) tree 인 dictree 를 이용하여 사전에 있는 단어들을 관리하는 프로그램을 작성해 보자. 참고로 사전의 각 단어들은 소문자 알파벳 (a-z) 로만 이루어져 있으며, 각 단어의 최대 길이는 MAX_SIZE 로 정의되어 있다. 이제 dictree를 이용하여 사전에 있는 단어들을 어떻게 관리하는지 알아보자.

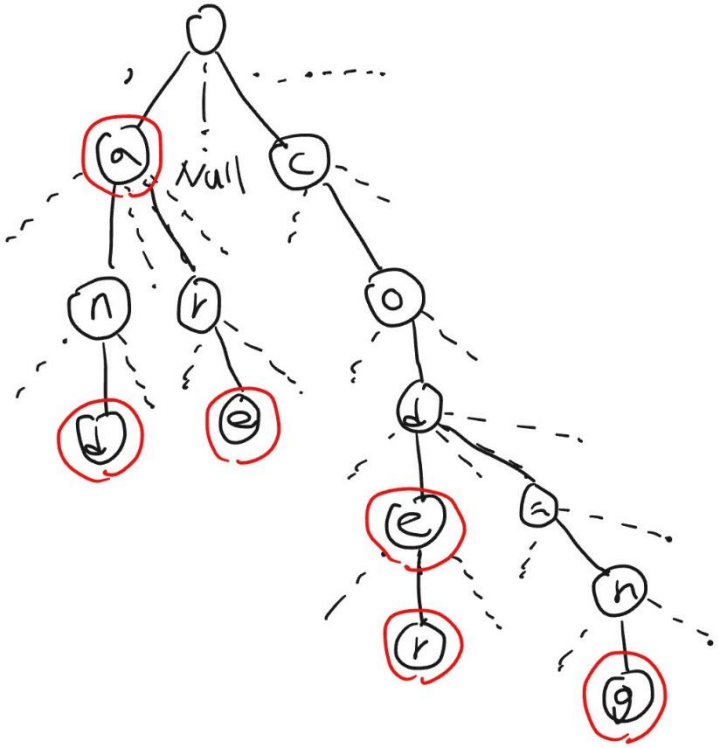

그림 1 dictree 예제 (점선 부분은 다른 알파벳에 대응하는 child node 들로서 모두 NULL 이다)

- Root node 는 아무런 알파벳도 가지고 있지 않는다. 나머지 node 들은 node 마다 **알파벳** 하나를 포함하고 있다 (struct dictree 참조)

- 임의의 dictree 상의 node k 에 대하여 k 의 **k-th (k 는 0 에서 25 사이) child 는 k 번째 알파벳을 가지고 있다**. 아래 그림 1 예시에서 root node 의 0th (즉 제일 왼쪽) node 는 알파벳 a 를 포함하고, 2nd (세번째) child 는 알파벳 c 를 포함한다. Ordered tree 이므로 일부 순서에 해당하는 child 가 물론 존재하지 않을 수도 있으며, 이 경우 해당 child node 는 NULL 이 된다.

- 이제 dictree 에서 사전의 각 단어들이 어떻게 관리되는지 알아보자. 각 node 는 알파벳 외에 추가적으로 **special 이라고 하는 값을 가지며**, 이 값은 0 또는 1 이다. 이때 어떤 node k 의 special 값이 1이면 node k 는 special node 가 되며, **사전은 dictree 의 root 에서 special node 까지의 path 가 해당하는 알파벳을 순서대로 이어 붙인 단어 모두를 포함한다**. 예를 들어 그림 1 의 경우, 붉은색 원으로 둘러 쌓인 node 가 special node 일 때, dictree 가 포함하는 사전의 단어들은 a, ad, are, code, coder, coding 이 되며, 이 외에 다른 단어는 가지고 있지 않는다. Root node 는 special node 가 될 수 없으며, root node 를 제외한 **dictree 의 leaf node 는 언제나 special node 가 되도록 관리한다**.

정리하면 dictree 의 각 node 는 다음과 같은 구조체로 관리할 수 있다 (이 경우 dictree 는 수업 시간에 배운 binary tree 와 마찬가지로 (root) node 의 pointer 형태로 선언하여 생성 및 관리할 수 있다).

```
struct dictree
{
    int special;                    // 1 if the node is special
    struct dictree* character[26];   //character[i] = NULL if there is no i-th child
}dctree;
```

이제 dictree 에 단어가 어떻게 추가되고 지워지는지 알아보자.

Case 1. 단어 S = s_1 s_2 … s_k 를 dictree 에 추가할 경우.

1. dictree 의 root node 에 s_1 에 해당하는 child node 가 있는지 검사, 있으면 해당 child node 로 이동하고, 없으면 새로 만들어 준 다음에 이동.

2. 과정 1을 s_k 에 해당하는 node 까지 이동하며 반복. 마지막까지 이동 후 해당 node 를 special node 로 만들어 줌.

예1) 그림 1의 dictree 에서 단어 'bee' 를 추가할 경우, root node 에서 알파벳 b에 해당하는 node 가 없으므로 새로 만들어준 뒤 이동한다. 나머지 두 ee node 를 같은 방법으로 만들어 준 뒤, 알파벳 e 를 가지고 있는 새로운 leaf node (bee 의 두번째 e 에 대응) 의 special 값을 1로 만들어 준다.

예2) 그림 1의 dictree 에서 단어 'cod' 를 추가할 경우, root->c->o->d 로의 path 가 이미 존재하므로 추가적인 node 의 추가 없이 알파벳 d 에 해당하는 node 를 special 로 만들어 준다.

Case 2. 단어 S = s_1 s_2 … s_k 를 dictree 에서 제거할 경우.

0. 단어 S가 현재 dictree 에 존재하지 않는 경우 변경 없음.

1. 아닌 경우, Case 1 과 마찬가지로 s_k 에 해당하는 node 까지 이동 (이동할 수 없다는 것은 해당 단어는 현재 dictree 가 포함하지 않는 다는 것을 의미). 해당 node 를 k 라 하자

1-1. Node k 가 leaf node 가 아니라면, 해당 node 가 더 이상 special node 가 아님을 나타내기 위해 special 값을 0으로 만들고 종료.

1-2. Node k 가 leaf node 라면 해당 node를 삭제. 삭제 후 삭제한 node 의 parent node 가 special node 가 아닌 leaf node 가 된다면, 해당 parent 도 삭제.

1-3. 1-2 과정을 leaf 에서 root node 로 올라가면서 더 이상 삭제를 안할 때까지 반복. 삭제를 안하는 node 가 처음으로 나오는 순간 종료.

예0) 그림 1의 dictree 에서 단어 'cod' 를 삭제할 경우, 우선 root->c->o->d 로 이동한다. 알파벳 d 에 해당하는 단어가 special 이 아니므로 해당 단어는 dictree 가 포함하지 않는다. 따라서 아무것도 하지 않는다.

예1) 그림 1의 dictree 에서 단어 'code' 를 삭제할 경우, 우선 root->c->o->d->e 로 이동한다. 알파벳 e 에 해당하는 special node 이지만 leaf node 가 아니므로 해당 node 의 special 값을 0으로 만들고 종료한다.

예2) 그림 1의 dictree 에서 단어 'coding' 을 삭제할 경우, 우선 root->c->o->d->i->n->g 로 이

동한 다음 알파벳 g에 해당하는 node 가 leaf node 이므로 해당 node 를 삭제한다. 삭제 뒤, g 의 parent node (알파벳 n에 대응) 또한 special node 가 아닌 leaf node 가 되므로 해당 node 또한 삭제한다. 알파벳 i 에 해당하는 node 도 마찬가지로 삭제되며, 그 다음 알파벳 d 에 해당하는 node 는 앞선 node 들이 모두 삭제된 뒤에도 여전히 leaf node 가 아니기 때문에 삭제되지 않고 종료된다.

- 에러 발생시 에러 메시지를 출력하는 print_error() 함수가 미리 구현되어 있다.

- 마지막으로 테스트를 위해 dic.c 에 현재 dictree 가 포함하고 있는 단어들을 모두 출력하는 **void printall (dictree * dic, char word[MAX_SIZE])** 가 구현되어 있으며, 호출 시 word 에는 빈 문자가 들어가야 한다 (실제 사용 예시는 main.c 를 통해 확인해 보자).

2. 구현해야 하는 함수들

2-1 dictree * construct (dictree * dic, char * file) : file 에 있는 정보를 읽어 와서 file에 있는 단어들을 모두 포함하는 dictree 를 하나 생성 후 반환한다. file 에서 각 단어는 줄바꿈 문자로 구분되어 있으며 (sample.txt 참고), 올바르지 않은 input 에 대한 error 처리는 하지 않아도 무방하다.

2-2 int search (dictree * dic, char * word) : 현재 dictree dic 이 word 단어로 포함 하고 있는 경우 1을, 그렇지 않은 경우 0을 반환한다.

2-3 int prefixsearch (dictree * dic, char * pre_word) : 현재 dictree dic 이 pre_word 를 prefix (전치사) 로 가지고 있는 단어를 포함하고 있는 경우 1을, 그렇지 않은 경우 0을 반환한다. 여기서 어떤 **string S의 prefix 는 S 의 처음 위치부터 임의의 위치 까지로 구성된 S의 substring 으로 정의된다**. 예를 들어 coding 의 prefix 는 c, co, cod, codi, codin, coding 이다.

2-4 int count_node (dictree * dic, int special) : special 이 0인 경우 현재 dictree dic 이 가지고 있는 non-special node 수를 (non-special node 에는 root node 도 포함됨에 유의하자), 1인 경우 dictree dic 이 포함하고 있는 special node 수 (= 단어 수) 를 출력한다. special 에 1, 0 외의 값이 들어올 경우 print_error() 를 호출한다.

2-5 dictree *insert (dictree *dic, char *word) : 현재 dictree dic 에 word 를 추가 후, 변경된 dictree 를 (즉 dictree 의 root node 를) 반환한다 (tip : insert 를 먼저 올바르게 구현할 경우, 앞의 construct 를 쉽게 구현할 수 있음).

2-6 dictree *delete (dictree *dic, char *word) : 현재 dictree dic 에 word 를 제거 후, 변경된 dictree 를 반환한다 (hint : recursion 을 이용하여 구현).

3. 주의 **(이 중 하나라도 어기면 0점)**

- 주어진 파일은 다음과 같은 형식으로 되어 있음

1. dictree.h : dictree.c 에서 정의한 함수 및 구조체들을 모아 둔 헤더파일.

2. dictree.c : dictree.h 에서 정의한 함수들의 세부 구현 (숙제에서 제출해야 할 파일).

3. main.c : 테스트용 main file (결과값이 주석에서 언급한 답과 일치하는지 확인할 것.

4. Makefile : 리눅스, 맥 환경에서 작업하는 학생들을 위한 Makefile.

5. sample.txt : 테스트 용 sample file

- 숙제 제출은 due 전까지 e-campus의 과제탭의 **과제물 제출 – 파일 첨부를 이용**하여, 오직 **dictree.c 파일만을 학번_dictree.c 파일로 업로드** 할 것. (예 : 2020000000_dictree.c). **그 외 어떠한 파일도 받지 않음**.

- 숙제의 delay는 받지 않음.

- 문제의 skeleton code 에서 주어진 헤더 파일 외의 **어떠한 헤더 파일도 include 하지 말 것.**

- 주어진 코드에 정의된 함수 및 전역 변수만을 구현 및 사용 하고, 자기가 따로 함수나 전역변수를 정의하여 dictree.c 에 추가하지 말 것 (채점 시 dictree.c 제외 원래 주어진 파일들로 채점한다는 것을 잊지 말 것). 또한 모든 C 함수는 반드시 표준 C함수 (standard C) 함수 및 constant 들만

사용할 것 (Visual studio 에서만 정의된 C 함수 및 라이브러리를 사용시 0점).

Standard C 헤더 파일 및 함수, constant 목록은 (https://en.wikipedia.org/wiki/C_standard_library) 에서 확인 가능

- 채점 시 GCC 6.3.0 (컴파일 시 과제에서 주어진 Makefile 이용), 및 Visual studio 2019 에서 채점할 예정이며, 둘 중 한곳에서 컴파일이 안될 시 무조건 0점.

- 제출하기 전에 예제로 주어진 main.c 파일을 컴파일 및 실행 뒤 실제로 올바르게 출력되는지 확인할 것 (올바른 결과값은 main.c 주석에 언급되어 있음).