

**## 스펙이 상당히 복잡하므로 미리 충분히 읽고 이해하기 바랍니다. 문서에 적혀 있는 부분에 대해서 제대로 이해하지 않고 하는 질문은 받지 않겠습니다.**

## 1. 개요

Hash-table, AVL tree 등 여러 자료 구조를 혼합해서 사용하는 법을 익힙니다.

## 2. 뼈대 코드

이 코드를 기본으로 하여 내용을 추가하도록 합니다.

이 코드에는 제출을 위한 입출력과 파일이름만이 정의되어 있습니다.

## 3. 지원하는 명령어

명령어는 첫 글자로 구분되며, 명령어의 종류를 나타내는 기호와 명령어의 내용은 하나의 공백 문자로 구분됩니다. 자세한 구현 방식은 **5. 세부사항** 부분을 참고해 주세요.

### 1. 데이터 입력: < (FILENAME)

패턴을 검색할 텍스트 파일을 입력합니다. 절대경로 및 상대경로를 모두 입력할 수 있습니다. 파일 이름 및 경로에는 공백이 포함되지 않는다고 가정해도 좋습니다. 새로운 텍스트 파일이 입력된 경우 이전에 입력된 데이터는 지워집니다.

### 2. 저장된 데이터 출력: @ (INDEX NUMBER)

입력한 번호에 해당하는 hash table 의 slot 에 담긴 문자열을 출력합니다. 전위 순회(preorder traversal) 방식으로 트리의 모든 노드를 출력합니다. slot 이 비어 있을 경우 EMPTY 를 출력합니다.

### 3. 패턴 검색: ? (PATTERN)

데이터 파일에서 등장하는 패턴의 위치를 모두 출력합니다. 패턴 문자열의 길이는 6 이상입니다.

패턴에 등장할 수 있는 문자의 종류에는 제한이 없습니다. 즉, 공백 문자도 패턴에 포함될 수 있습니다.

패턴의 위치는 (줄 번호, 시작 글자의 위치)의 형식으로 출력합니다. 첫 번째

줄의 첫 번째 글자는 (1, 1)입니다.

패턴이 여러 번 등장하는 경우 각 좌표는 공백으로 구분합니다. 마지막 좌표 뒤에는 공백을 출력하지 않습니다.

패턴이 검색되지 않는 경우 (0, 0)을 출력합니다.

#### 4. 종료: QUIT

프로그램을 종료합니다.

## 4. 입력

입력 데이터는 한 줄에 문자열이 하나씩 있는 파일로 받습니다. 여러분은 파일을 읽으면서 총 줄 수를 세어야 합니다. 모든 줄은 6 글자 이상으로 이루어져 있습니다.

이 예제는 5 줄이므로 string 수는 5 개입니다.

```
this is a boy. hello, boy.  
it is more important to avoid using a bad data structure.  
i am a boyboy. boys be ambitious!  
boyboyoboyboyboy  
more important to avoid it is more important to data
```

## 5. 세부 사항 **반드시 여기에 맞는 방식으로 구현해야 합니다.**

전체 string 의 개수가  $n$  일 때, string 의 집합을  $S = \{ S_1, S_2, \dots, S_n \}$  이라고 하자.

$S$  로 아래에 제시된 hash table 과 AVL tree, linked list 를 구성한다.

string  $S_i$  ( $i=1,2, \dots, n$ )의 길이가  $m$  일 때,  $S_i$ 에 대해서 길이  $k$  인 substring  $S_i[1..k]$ ,  $S_i[2..k+1]$ , ...,  $S_i[m-k+1..m]$  이 존재한다. ( $S_i[x..y]$ 는 index 가  $x$  부터  $y$  까지인 substring,  $1 \leq x \leq y \leq m$ )

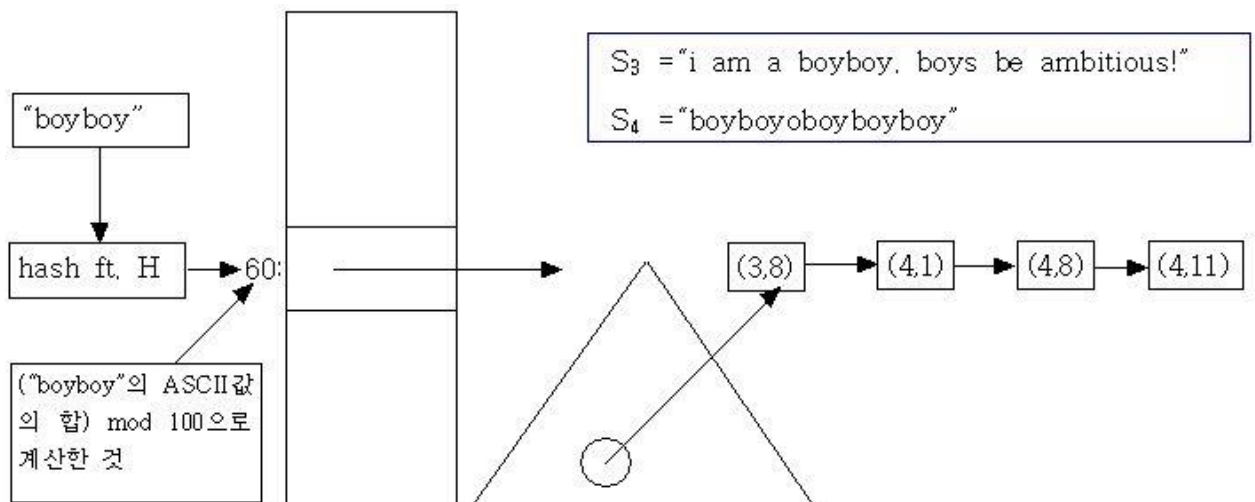
길이  $k$  인 모든 substring  $S_i[j..j+k-1]$  ( $1 \leq i \leq n$ ,  $1 \leq j \leq m-k+1$ ,  $m$ :  $S_i$ 의 길이) 에 대하여 아래의 과정을 수행한다.

- 각 substring 을 hashing 한다.

1. hash function: ( $k$  character 들의 ASCII code 들의 합) mod 100

2. table 의 크기는 collision 을 유발하기 위해 비현실적이지만 100 으로 한다.

- hash table 의 각 slot 은 AVL tree 로 구현한다. 서로 다른 substring 이지만 hashing 값이 같으면 collision 이 일어나므로 이들은 AVL tree 로 구별한다.
- AVL tree 에서 노드의 추가로 인해 높이의 불균형이 발생하였을 경우, 해당하는 노드 중 새로 추가된 노드에서 가장 가까운 조상을 기준으로 회전 연산을 적용한다.
- AVL tree 의 각 node 는 linked list 로 구현한다. 하나의 substring 이 S 상에서 여러 번 등장할 수 있다. 이러한 경우 AVL tree 의 해당 node 에서 linked list 로 연결, 관리한다.
- string 이  $S[i..j+k-1]$  일 때, linked list 의 node 는  $(i, j)$  값을 갖는다.
- 예를 들어,  $S_3 = \text{"i am a boyboy. boys be ambitious!"}$ ,  $S_4 = \text{"boyboyoboyboyboy"}$ ,  $k = 6$  이고,  $H(\text{"boyboy"})$ 에 의해 결정된 hash table 상의 index 가 ( $\text{"boyboy"}$ 의 ASCII 값의 합) mod 100 이면, substring  $S_3[8..13]$ ,  $S_4[1..6]$ ,  $S_4[8..13]$ ,  $S_4[11..16]$ 에 의해 아래 그림과 같은 자료구조가 생성된다.
- S 상에서 길이 k 인 모든 substring 에 대해 그림과 같이 구성한다.  
본 숙제에서  $k = 6$  으로 잡는다.  
string 의 길이가 6 미만인 경우는 구현의 단순성을 위해 고려하지 않는다.  
모든 실제 문제에서 사용되는 data string, pattern string 의 길이는 6 이상이고, 이를 처리하기 위한 자료구조에서 검색의 key 가 되는 substring 의 길이는 6 으로 고정한다.  
즉, 길이가 6 인 substring 들에 대한 저장된 정보를 바탕으로 길이가 6 이상인 패턴 스트링을 처리하는 것이다.



## 7. 예제 입출력

```
$ java Matching
< data.txt
? boyboy
(3, 8) (4, 1) (4, 8) (4, 11)
? important to
(2, 12) (5, 6) (5, 36)
? algorithm
(0, 0)
@ 60
portan boyboy oyboyb yboyob yboybo yoboyb
@ 73
oyboyoy ata st re imp oyoboy
@ 0
EMPTY
QUIT
$
```

```
← 프로그램 실행
← 이렇게 입력
← 이렇게 입력하면
← 이렇게 출력한다.
← 이렇게 입력하면
← 이렇게 출력한다.
← 이렇게 입력하면
← 이렇게 출력한다.
← 이렇게 입력하면
← 이렇게 출력한다.
← 이렇게 입력하면
← 이렇게 출력한다.
← 이렇게 입력하면
← 이렇게 출력한다.
← 이렇게 입력하면
← 종료한다.
```

## 8. 참고사항

1. 이 과제의 핵심 부분은 복잡한 자료구조의 구현입니다. AVL 트리는 직접 구현해야 하며 다른 코드를 가져오는 것을 금합니다. (**public domain 포함**)
2. 제출 전, 반드시 컴파일 가능 유무와 프로그램의 수행시간을 확인합니다. 컴파일시 옵티마이제이션 옵션은 쓰지 않습니다.

## 9. 자주 묻는 질문(필독!!!)

### 1)API 사용

AVL Tree 는 public domain 포함하여 일체의 코드도 가져오지 않고 본인이 직접 구현해야합니다.

(일반적인 Tree 의 구성요소는 public domain 과 비슷해질 수도 있으나 AVL Tree 가 갖는 특성은 본인의 코드여야 합니다.)

Hashtable 과 LinkedList 는 API 사용을 허용하기는 하나 수업시간에 나오는 스펙을 모두 만족해야 합니다.

## 2)입력

명령어의 종류를 나타내는 문자(<,@,?)와 명령입력 사이는 ' ' (space) 1 개로만 이루어져 있습니다.

과제에 명시된 형태의 입력만 들어온다고 가정해도 좋습니다.

@ 명령은 0~99 사이의 index 만 들어온다고 가정해도 좋습니다.

## 3)출력

해당 패턴이 등장하는 좌표를 출력할 때에는 정렬된 순서로 출력하시면 됩니다.

모든 출력의 맨 끝에는 정해진 정답 이외에 뒤따라 붙는 공백이 없습니다.

## 4)AVL Tree

AVL Tree 의 Key 는 길이 k 의 Substring 이거나 각 Substring 을 구분할 수 있는 형태이면 되고, String 을 사전 순으로 비교할 수 있으면 됩니다.

## 5)구현

Hashtable, LinkedList, AVL Tree 를 Generic 으로 구현할 경우 그렇지 않은 경우보다 코드점수를 좋게 받을 수 있습니다. 반드시 점수가 높은 것은 아니지만 비슷한 구현>의 경우 Generic 구현이 좀 더 점수가 높을 수 있습니다. 그래도 Generic 이 얻는 이득보다 Testcase 를 많이 맞추는 것이 더 이득이므로 정확하게 구현해주세요.

첫 시작 6 글자만 자료구조 안에서 찾은 뒤 해당 좌표에 해당하는 부분을 다시 파일을 읽어서 패턴과 비교하는 식의 구현은 안됩니다.