프로그래밍 과제 06

진짜 처럼 보이는 그럴 듯한 가짜 랜덤 텍스트를 만들어내는 방법 중의 하나로 Markov chain을 이용하는 방법이 있다. 이 방법에서는 먼저 하나의 진짜 텍스트가 주어진다. 예를 들어 다음과 같은 진짜 텍스트를 예로 들어보자.

```
Show your flowcharts and conceal your tables and I will be
mystified. Show your tables and your flowcharts will be
obvious. [end]
```

이 텍스트에서 모든 "연속된 두 단어의 쌍"과 "그 쌍에 이어서 바로 등장한 단어"의 목록을 찾아낸다. 연속된 두 단어의 쌍을 prefix라고 부르고, "이어서 등장한 단어"를 suffix라고 부르자.

PREFIX	SUFFIX
Show your	flowcharts(1), tables(1)
your flowcharts	and(1), will(1)
flowcharts and	conceal(1)
and conceal	your(1)
flowcharts will	be(1)
your tables	and(2)
will be	mystified.(1), obvious.(1)
be mystified.	Show(1)
be obvious.	[end]
tables and	I(1), your(1)
and I	will(1)
…	

예를 들어 Show your라는 prefix 다음에는 flowcharts라는 suffix와 tables라는 suffix가 각각 1번씩 등장한 적이 있다. 그리고 your tables라는 prefix 다음에는 and가 2번 등장한 적이 있다. 즉 괄호 안의 숫자는 그 suffix가 해당 prefix 바로 다음에 등장한 횟수를 표시한다. 마침표, 쉼표 등의 기호는 단어의 일부로 간주하고, 대소문자는 구분한다. 그리고[end]는 텍스트의 끝을 의미한다. 이제 랜덤 텍스트를 만드는 Markov chain 알고리즘은 다음과 같이 진행된다. 먼저 텍스트를 시작할 두 단어를 입력 받는다. 예를 들어 두 단어가 Show와 your라고 가정하자. 그러면 먼저 Show your를 출력한다.

```
Show your
```

그런 다음 Show your의 suffix인 flowcharts와 tables 중의 하나를 1/2의 확률로 선택한다. 가령 tables가 선택되었다고 가정하자. 그럼 tables를 출력하고

```
Show your tables
```

이제 가장 마지막으로 출력된 두 단어인 your tables가 prefix가 된다. 이 prefix에 대한 suffix는 and로 유일하므로 and를 출력한다.

```
Show your tables and
```

이제 tables and가 prefix가 되고 suffix인 I와 your 중에 하나를 1/2의 확률로 선택한다. 일반적으로는 suffix들을 등장 빈도에 비례하는 확률로 선택해야 한다. 가령 I의 등장 빈도가 3이고 your가 1이었다면 3/4의 확률로 I를, 1/4의 확률로 your를 선택해야 한다. 만약 I가 선택되었다면 I를 출력하고

```
Show your tables and I
```

이제 and I의 유일한 suffix인 will을 출력한다.

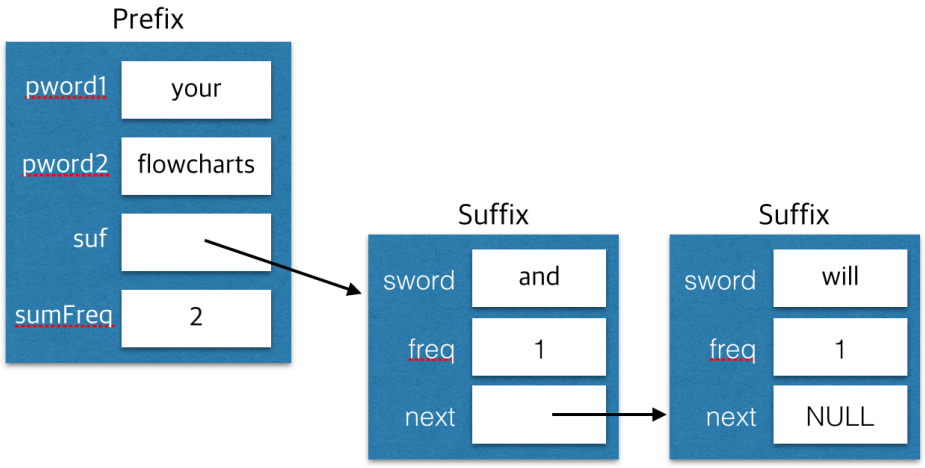
```
Show your tables and I will
```

이런 식으로 계속하여 더이상 suffix가 존재하지 않거나 (즉 [end]에 도달하거나) 혹은 생성할 랜덤 텍스트의 길이의 최대값에 도달하면 종료한다.

입력으로 주어진 진짜 텍스트를 분석하여 찾아낸 모든 prefix들과 suffix들, 즉 위의 테이블의 내용을 저장할 자료구조를 정의해 보자. 하나의 prefix가 여러 개의 suffix들을 거느리고 있는 형태이므로 이것을 다음과 같은 구조체로 표현한다.

```
typedef struct suffix Suffix;
struct suffix {
    char *sword;              // suffix인 단어
    int freq;                 // 등장횟수
    Suffix *next;             // 다음 노드의 주소
};

typedef struct prefix {
    char *pword1;             // prefix를 구성하는 첫 단어
    char *pword2;             // prefix를 구성하는 두 번째 단어
    Suffix *suf;              // suffix들의 연결리스트의 첫 번째 노드의 주소
    int sumFreq;              // suffix들의 등장횟수(freq)의 합
} Prefix;
```

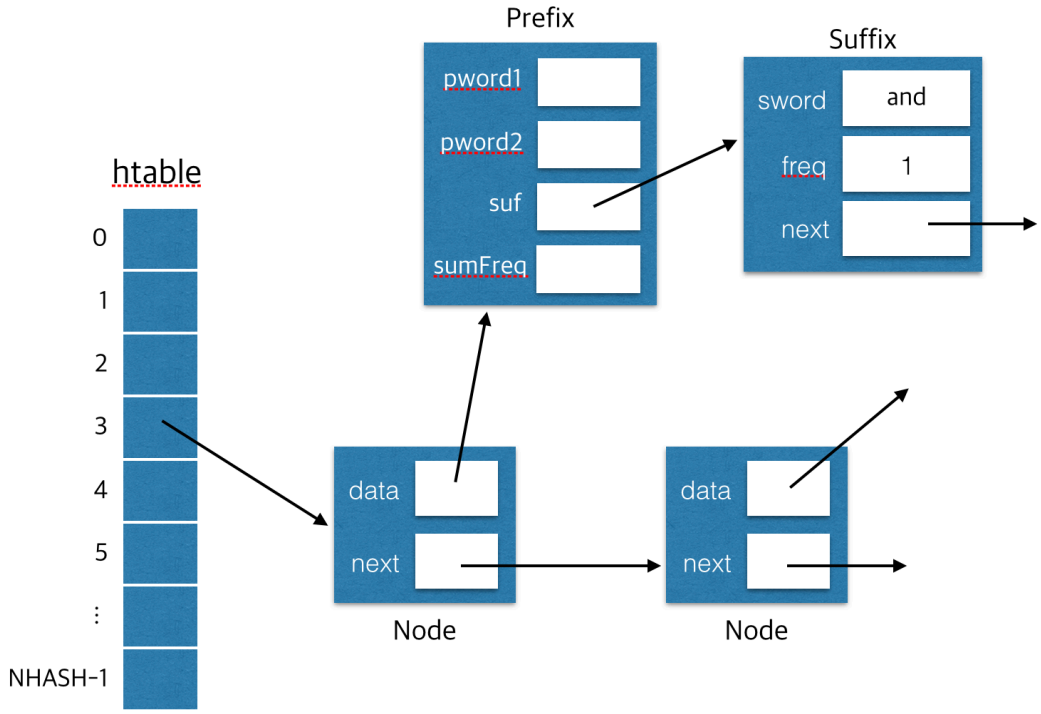


위의 그림과 같은 구조를 가지는 여러 개의 Prefix객체들을 저장하기 위해서 해쉬 테이블을 사용한다. 충돌 해결은 chaining기법을 사용한다. 즉, 해쉬 테이블의 각 칸은 하나의 연결리스트를 거느리고 있고, 그 연결리스트의 각 노드에는 하나의 Prefix 객체의 주소를 저장한다.

```
#define NHASH 4093            // NHASH는 해쉬 테이블의 크기
typedef struct node Node;
```

```
typedef struct node {
    Prefix *data;
    Node *next;
};

Node *htable[NHASH];          // 해쉬 테이블
```



해쉬함수는 Prefix를 구성하는 두 단어를 키(key)로 사용한다. 다음의 해쉬 함수를 이용하라.

```
#define MULTIPLIER 31
unsigned int hash(char *key1, char *key2) {
  unsigned int h = 0;
  unsigned char *p;

  for (p = (unsigned char *)key1; *p != '\0'; p++)
      h = MULTIPLIER * h + *p;
  for (p = (unsigned char *)key2; *p != '\0'; p++)
      h = MULTIPLIER * h + *p;
  return h % NHASH;
}
```

입력으로 주어진 진짜 텍스트 파일은 다음의 샘플 파일을 이용하고, 가짜 텍스트의 최대 길이는 1000단어로 하라.