프로그래밍 과제 04

이 과제는 C++, Java 혹은 Python으로 작성한다.

진짜 처럼 보이는 그럴 듯한 가짜 랜덤 텍스트를 만들어내는 방법 중의 하나로 Markov 체인을 이용하는 방법이 있다. 이 방법에서는 먼저 하나의 진짜 텍스트가 주어진다. 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 등장했다. 그리고 your tables라는 prefix 다음에는 and가 2번 등장했다. 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 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. 원래는 suffix들을 등장 빈도에 비례하는 확률로 선택해야 한다. 가령 I의 등장 빈도가 3이고 your가 1이었다면 3/4의 확률로 I를, 1/4의 확률로 your를 선택해야 한다. 하지만 이 과제에서는 문제를 쉽게 만들기 위해서 그냥 suffix들 중에서 하나를 동일한 확률로 선택한다. 따라서 위의 테이블에서 처럼 각 suffix들의 등장 빈도를 저장할 필요는 없다. 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.

자료구조

이 문제를 해결하기 위해서는 key=prefix와 value=suffixes인 (key, value) 쌍들을 저장할 자료구조가 필요하다. 여기서 prefix는 2개의 단어로 구성되고, suffixes는 단어들의 집합 혹은 리스트이다. 일반적으로 (key, value) 쌍들을 저장하고, key에 대한 검색을 지원하며, (key, value)쌍의 삽입과 삭제 등의 연산을 지원하는 자료구조를 map 혹은 dictionary라고 부른다. 일반적으로 map과 dictionary는 레드-블랙 트리와 같은 검색트리 혹은 해쉬 테이블을 이용해서 구현한다.

C++

C++은 레드블랙 트리를 사용하는 std::map과 해쉬 테이블을 사용하는 std::unordered_map을 제공한다. 이 과제에서는 std::unordered_map을 사용한다. 우선 prefix를 정의하는 하나의 클래스 Prefix를 정의한다. 하나의 Prefix는 2개의 단어로 구성된다. 특정한 Prefix에 대한 suffix들은 단어들의 집합이므로 하나의 vector<string>에 저장할 수 있을 것이다. 여기서 문제는 std::unordered_map이 기본적으로는 빌트인(built-in) 데이터 타입만을 키(key)로 허용한다는 것이다. Prefix와 같은 커스텀 클래스를 키로 사용하기 위해서는 약간의 추가적인 작업이 필요하다. 이에 대해서는 아래의 참조코드 혹은 여기 혹은 여기를 참조하라.

Java

Java API는 C++의 unordered_map에 대응하는 HashMap 클래스와 C++의 vector<string>에 해당하는 ArrayList<String>이 있다. 따라서 C++의 경우와 유사하게 할 수 있다.

Python

Python은 (key, value) 형식의 데이터를 저장하기 위한 빌트인 데이터 타입인 사전(dictionary)을 제공한다. 키(key)로는 prefix를 구성하는 두 단어의 튜플(tuple)을 사용하고, 값(value)으로는 suffix들의 리스트를 사용하면 된다. Python에서 리스트는 딕셔너리의 키로 사용될 수 없지만 튜플은 사용될 수 있다.

입력으로 주어진 진짜 텍스트 파일은 다음의 샘플 파일을 이용하고, 가짜 텍스트의 최대 길이는 100단어로 하라.

[참조 코드]

```
// CPP program to demonstrate working of unordered_map for user defined data types.
#include <bits/stdc++.h>
using namespace std;

struct Person {
```

```
	string first, last;

	Person(string f, string l)
	{
		first = f;
		last = l;
	}

	bool operator==(const Person& p) const
	{
		return first == p.first && last == p.last;
	}
};

class MyHashFunction {
public:

	// We use predefined hash functions of strings
	// and define our hash function as XOR of the
	// hash values.
	size_t operator()(const Person& p) const
	{
		return (hash<string>()(p.first)) ^
			(hash<string>()(p.last));
	}
};

// Driver code
int main()
{
	unordered_map<Person, int, MyHashFunction> um;
	Person p1("kartik", "kapoor");
	Person p2("Ram", "Singh");
	Person p3("Laxman", "Prasad");

	um[p1] = 100;
	um[p2] = 200;
	um[p3] = 100;

	for (auto e : um) {
		cout << "[" << e.first.first << ", "
			<< e.first.last
			<< "] = > " << e.second << '\n';
	}

	return 0;
}
```