**21. Алгоритм Ахо-Карасик (бор)**

**Алгоритм Ахо-Корасик**: Пусть дан набор строк в алфавите размера k суммарной длины m. Алгоритм Ахо-Корасик строит для этого набора строк структуру данных "бор", а затем по этому бору строит автомат, всё за O(m) времени и O(mk) памяти. Полученный автомат уже может использоваться в различных задачах, простейшая из которых — это нахождение всех вхождений каждой строки из данного набора в некоторый текст за линейное время.

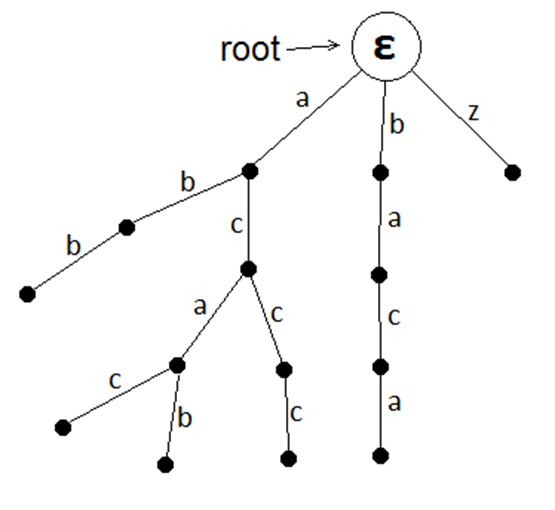
Данный алгоритм был предложен канадским учёным Альфредом Ахо (Alfred Vaino Aho) и учёным Маргарет Корасик (Margaret John Corasick) в 1975 г.

**Построение бора:** Что же такое бор? Строго говоря, бор – это дерево, в котором каждая вершина обозначает какую-то строку (корень обозначает нулевую строку – ε).

На ребрах между вершинами написана одна буква (в этом его принципиальное различие с суффиксными деревьями и др.), таким образом, добираясь по ребрам из корня в какую-нибудь вершину и конкатенируя буквы из ребер в порядке обхода, мы получим строку, соответствующую этой вершине. Из определения бора как дерева вытекает также единственность пути между корнем и любой вершиной, следовательно — каждой вершине соответствует ровно одна строка (в дальнейшем будем отождествлять вершину и строку, которую она обозначает).

Формально, **бор** — это дерево с корнем в некоторой вершине **Root**, причём каждое **ребро дерева** подписано некоторой буквой. Если мы рассмотрим список рёбер, выходящих из данной вершины (кроме ребра, ведущего в предка), то все рёбра должны иметь разные метки.

Вот пример построенного бора для строк: 1) acab, 2) accc, 3) acac, 4) baca, 5) abb, 6) z, 7) ac.



Рассмотрим в боре любой путь из корня; выпишем подряд метки рёбер этого пути. В результате мы получим некоторую строку, которая соответствует этому пути. Если же мы рассмотрим любую вершину бора, то ей поставим в соответствие строку, соответствующую пути из корня до этой вершины.

Каждая вершина бора также имеет флаг **leaf**, который равен **true**, если в этой вершине оканчивается какая-либо строка из данного набора.

Соответственно, **построить бор** по данному набору строк — значит построить такой бор, что каждой **leaf** – вершине будет соответствовать какая-либо строка из набора, и, наоборот, каждой строке из набора будет соответствовать какая-то **leaf** -вершина.

Опишем теперь, **как построить бор** по заданному набору строк за линейное время относительно их суммарной длины.

Введём структуру, соответствующую вершинам бора:

struct vertex {

int next[K];

bool leaf;

};

vertex t[NMAX+1];

int sz;

Т.е. мы будем хранить бор в виде массива (количество элементов в массиве - это sz) структур vertex. Структура vertex содержит флаг leaf, и рёбра в виде массива next[], где next[i]— указатель на вершину, в которую ведёт ребро по символу i, или -1, если такого ребра нет.

Вначале бор состоит только из одной вершины — корня (договоримся, что корень всегда имеет в массиве t индекс 0). Поэтому инициализация бора такова:

memset (t[0].next, 255, sizeof(t[0].next));

sz = 1;

Теперь реализуем функцию, которая будет добавлять в бор заданную строку s. Реализация крайне проста: мы встаём в корень бора, смотрим, есть ли из корня переход по букве s[0]: если переход есть, то просто переходим по нему в другую вершину, иначе создаём новую вершину и добавляем переход в эту вершину по букве s[0]. Затем мы, стоя в какой-то вершине, повторяем процесс для буквы s[1], и т.д. После окончания процесса помечаем последнюю посещённую вершину флагом leaf =true.

void add\_string (const string & s) {

int v = 0;

for (size\_t i=0; i<s.length(); ++i) {

char c = s[i]-'a'; // в зависимости от алфавита

if (t[v].next[c] == -1) {

memset (t[sz].next, 255, sizeof t[sz].next);

t[v].next[c] = sz++;

}

v = t[v].next[c];

}

t[v].leaf = true;

}

Линейное время работы, а также линейное количество вершин в боре очевидны. Поскольку на каждую вершину приходится O(k) памяти, то использование памяти есть O(nk).

**Задача**. Дан набор из n различных слов. Для каждого слова узнайте, сколько раз оно встречается как подстрока во всех остальных словах.

**Формат входных данных**

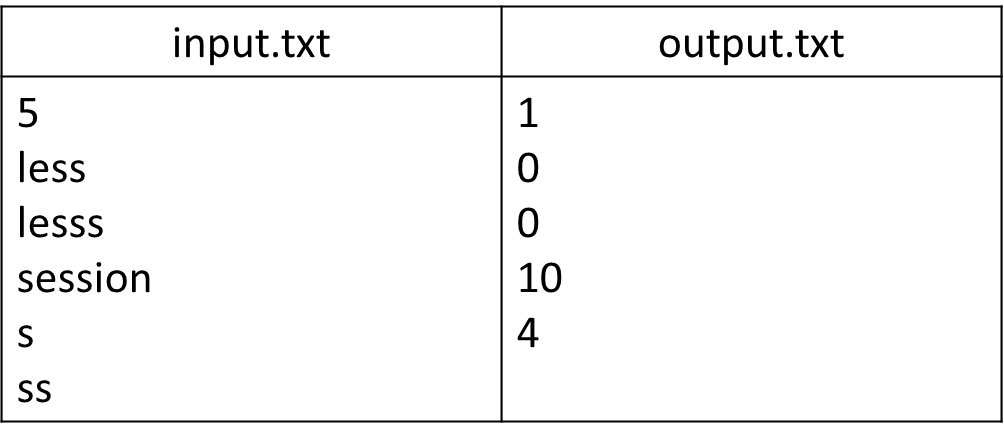
В первой строке входного файла записано целое число n (1≤n≤10000). В следующих n строках записаны слова. Каждое слово не пусто и состоит из не более чем 20 строчных букв латинского алфавита.

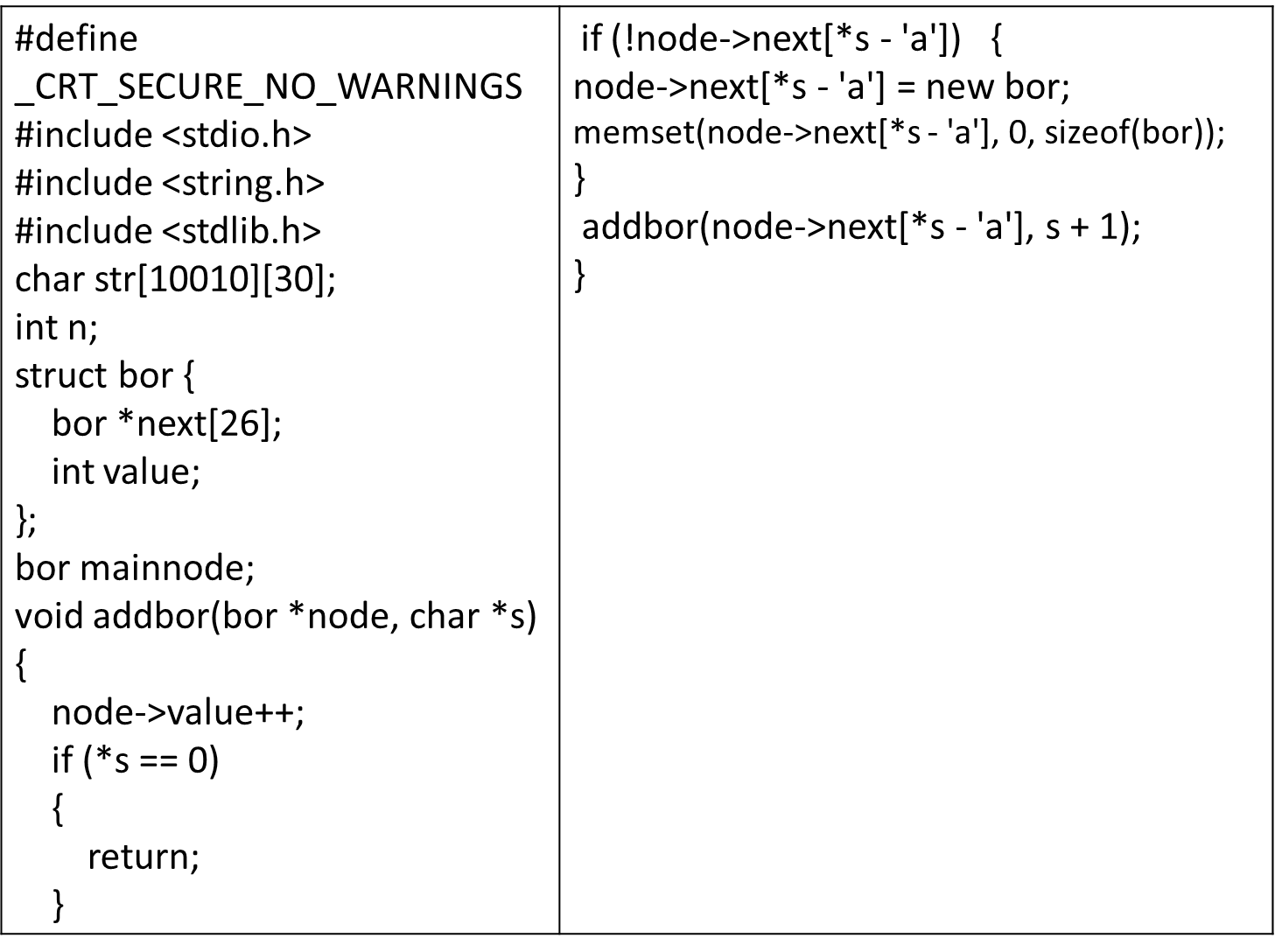
Все слова различны.

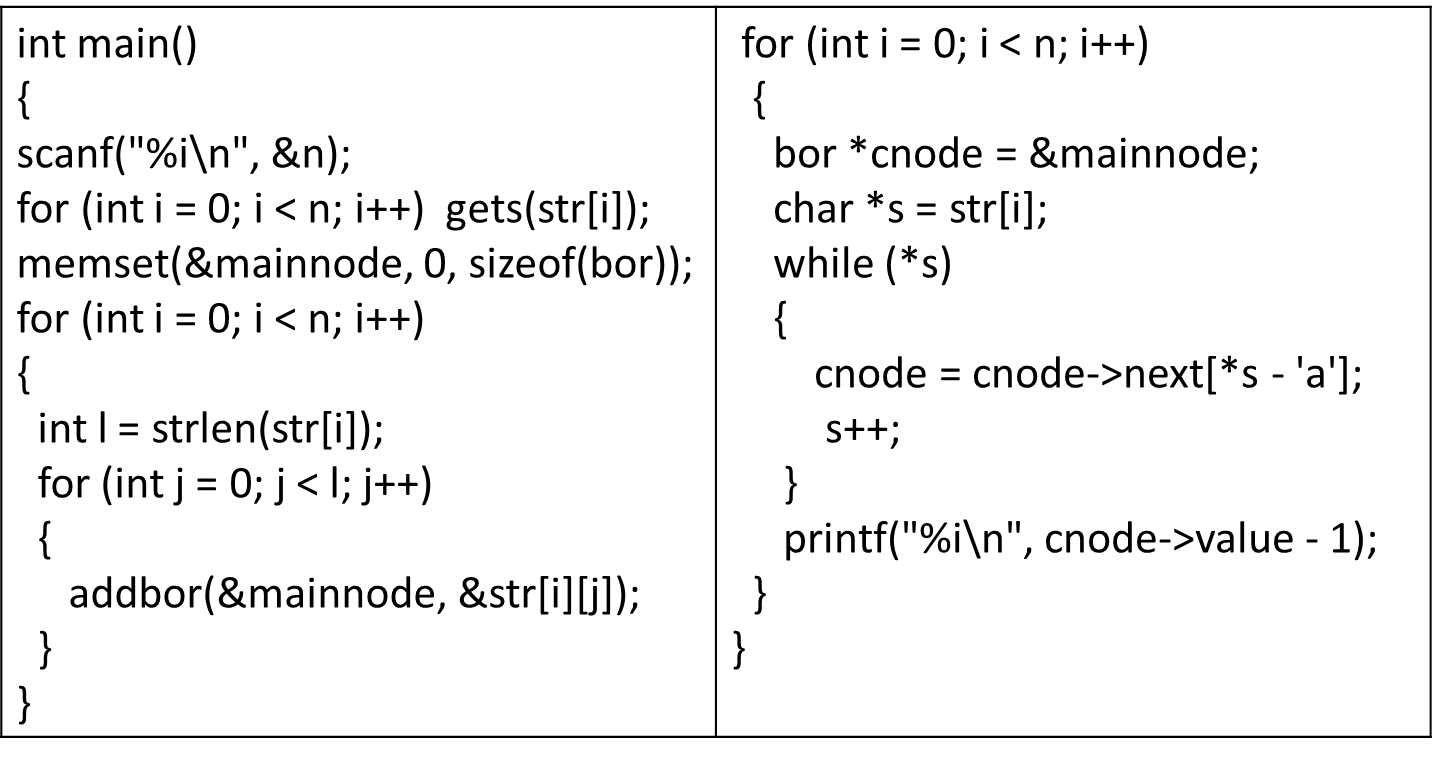
**Формат выходных данных**

В выходной файл выведите n строк, по одному числу на строке. В i-й строке должно быть записано, сколько раз i-е слово встречается в других словах как подстрока.

**Пример**

****

****

****