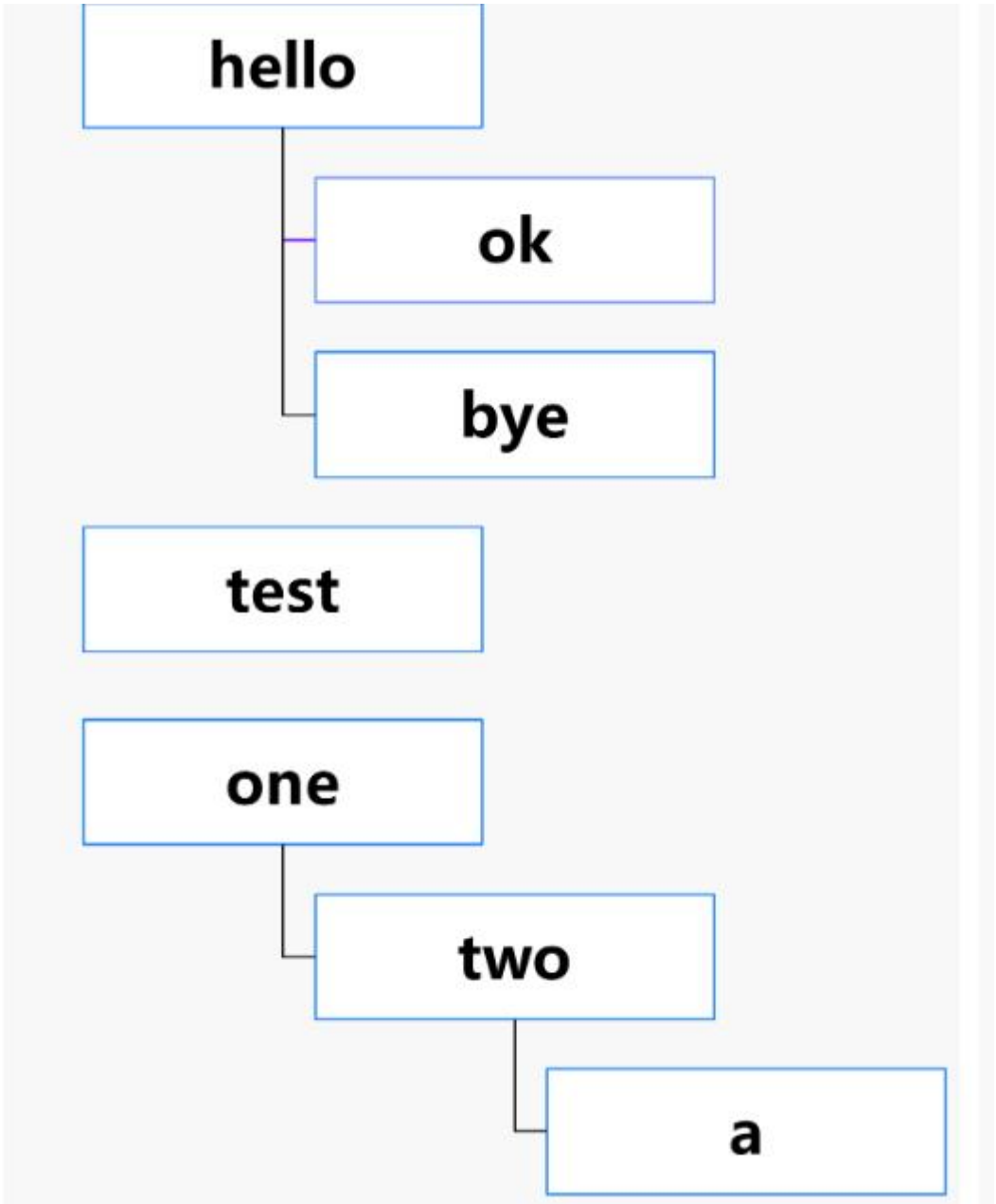
在一个博客网站上，每篇博客都有评论。每一条评论都是一个非空英文字母字符串。
评论具有树状结构，除了根评论外，每个评论都有一个父评论。
当评论保存时，使用以下格式：

- 首先是评论的内容；
- 然后是回复当前评论的数量。
- 最后是当前评论的所有子评论。（子评论使用相同的格式嵌套存储）

所有元素之间都用单个逗号分隔。
例如，如果评论如下：



第一条评论是"hello,2,ok,0,bye,0"，第二条评论是"test,0"，第三条评论是

"one,1,two,1,a,0"。

所有评论被保存成"hello,2,ok,0,bye,0,test,0,one,1,two,1,a,0"。

对于上述格式的评论，请以另外一种格式打印：
首先打印评论嵌套的最大深度。

然后是打印 n 行，第 i（$1<=i<=n$)行对应于嵌套级别为 i 的评论（根评论的嵌套级别为 1）。

对于第 i 行，嵌套级别为 i 的评论按照它们出现的顺序打印，用空格分隔开。

输入描述：

一行评论。由英文字母、数字和英文逗号组成。

保证每个评论都是由英文字符组成的非空字符串。

每个评论的数量都是整数（至少由一个数字组成）。

整个字符串的长度不超过 10^6。

给定的评论结构保证是合法的。

输出描述：

按照给定的格式打印评论。对于每一级嵌套，评论应该按照输入中的顺序打印。

补充说明：

输入：

```
hello,2,ok,0,bye,0,test,0,one,1,two,1,a,0
```

输出：

```
3

hello test one

ok bye two

a
```

说明：

如题目描述中图所示，最大嵌套级别为 3。嵌套级别为 1 的评论是"hello test one"，嵌套级别为 2 的评论是"ok bye two"，嵌套级别为 3 的评论为"a"。

示例 2

输入：

```
A,5,A,0,a,0,A,0,a,0,A,0
```

输出：

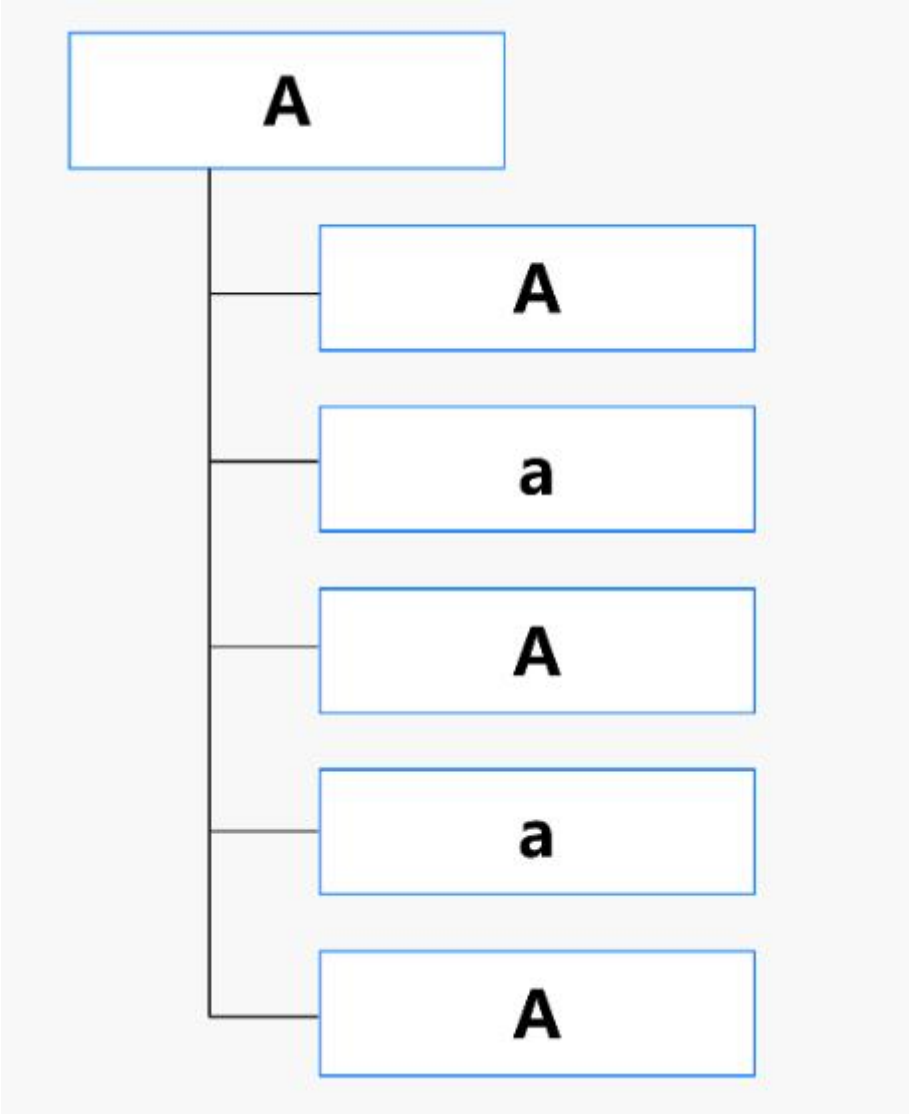
```
2

A

A a A a A
```

说明：

如下图所示，最大嵌套级别为 2，嵌套级别为 1 的评论是"A"，嵌套级别为 2 的评论是"A a A a A"



示例 3

输入：

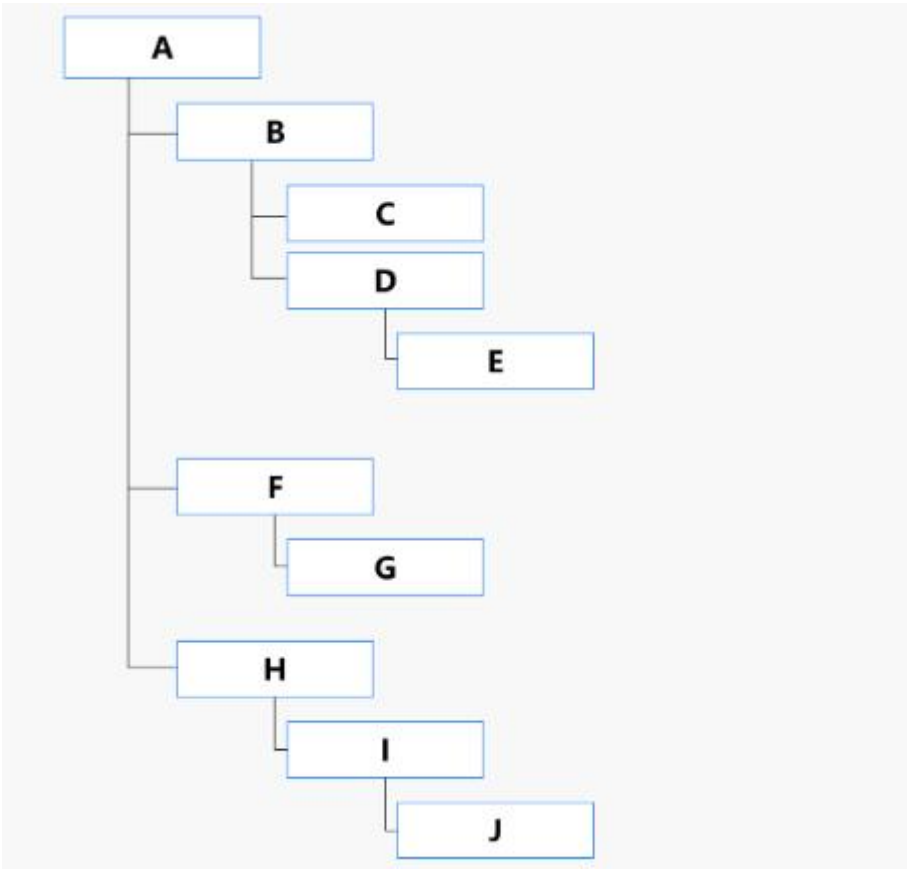
```
A,3,B,2,C,0,D,1,E,0,F,1,G,0,H,1,I,1,J,0,K,1,L,0,M,2,N,0,O,1,P,0
```

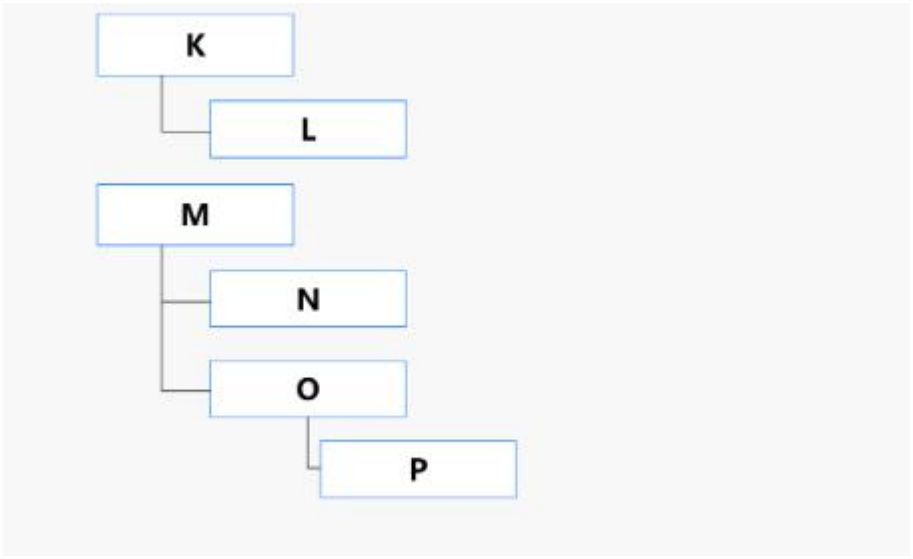
输出：

```
4
A K M
B F H L N O
C D G I P
E J
```

说明：

如下图所示。






```
#include <cstdio>
#include <cstdlib>
#include <vector>
#include <cstring>
#include <string>
#include <iostream>

using namespace std;
```

```
const int stringLen = 1024;

struct node
{
   char k[stringLen];
   int sunNum;
};

char *getNode(char *p, node &a)
{
   memset(&a, 0, sizeof(node));
   int i = 0;
   char tmpchar;
   for (int i = 0; i < stringLen; ++i)
   {
      tmpchar = *p++;
      if (tmpchar == ',')
      {
         break;
      }
      else
      {
         a.k[i] = tmpchar;
      }
   }
   if (i == stringLen)
   {
      printf("err\n");
   }
   for (int i = 0; i < stringLen; ++i)
   {
      tmpchar = *p++;
      if (tmpchar == ',' || tmpchar == 0)
      {
         break;
      }
      else
      {
         a.sunNum = a.sunNum * 10 + tmpchar - '0';
      }
   }
   return p;
}
```

```
char questionStr[1000100] = {};
vector<node> nodeVec;
vector<node>::iterator nodeVecIt;
vector<node>::iterator nodeVecEndIt;
vector<vector<string>> ansVec;
int maxLevel = 1;
int nowLevel = 1;

void dealNextLevelString(const node &faNode, int tmpLevel)
{
    if (faNode.sunNum == 0)
    {
        return;
    }
    ++tmpLevel;
    if (ansVec.size() < tmpLevel)
    {
        vector<string> tmpVec;
        ansVec.push_back(tmpVec);
    }
    for (int i = 0; i < faNode.sunNum; ++i)
    {
        node v = *nodeVecIt;
        ansVec[tmpLevel - 1].push_back(v.k);
        ++nodeVecIt;
        dealNextLevelString(v, tmpLevel);
    }
}

int main()
{
    scanf("%s", questionStr);
    char *p = questionStr;
    while (*p)
    {
        node a = {};
        p = getNode(p, a);
        nodeVec.push_back(a);
    }
    nodeVecIt = nodeVec.begin();
    nodeVecEndIt = nodeVec.end();
    if (nodeVec.size())
    {
        vector<string> tmpVec;
```

```
        ansVec.push_back(tmpVec);
    }
    while (nodeVecIt != nodeVecEndIt)
    {
        node v = *nodeVecIt;
        string tmpstr = v.k;
        ansVec[0].push_back(tmpstr);
        ++nodeVecIt;
        dealNextLevelString(v, 1);
    }
    int vecSize = ansVec.size();
    printf("%d\n", vecSize);
    for (int i = 0; i < vecSize; ++i)
    {
        for (int j = 0; j < ansVec[i].size(); ++j)
        {
            if (j)
            {
                printf(" ");
            }
            printf("%s", ansVec[i][j].c_str());
        }
        printf("\n");
    }
}
```