# 49、TLV 解析 II，考点 or 实现——字符串，数组，集合操作

## 题目描述

两端通过 TLV 格式的报文来通信，现在收到对端的一个TLV格式的消息包，要求生成匹配后的(tag, length, valueOffset)列表。

具体要求如下:

(1)消息包中多组tag、length、value紧密排列，其中tag,length各占1字节(uint8_t) , value所占字节数等于length的值

(2)结果数组中tag值已知，需要填充每个tag对应数据的length和valueOffset值(valueOffset为value在原消息包中的起始偏移量（从0开始，以字节为单位)),

即将消息包中的tag与结果数组中的tag进行匹配（可能存在匹配失败的情况，若结果数组中的tag在消息包中找不到，则length和valueOffset都为0)

(3)消息包和结果数组中的tag值都按升序排列，且不重复

(4)此消息包未被篡改，但尾部可能不完整，不完整的一组TLV请丢弃掉

## 输入描述

第一行: 一个字符串，代表收到的消息包。字符串长度在10000以内。

- 说明1: 字符串使用十六进制文本格式（字母为大写）来展示消息包的数据，如0F04ABABABAB代表一组TLV:前两个字符(0F）代表tag值为15，接下来两个字符（04）代表length值为4字节，接下来8个字符即为4字节的value。
- 说明2: 输入字符串中，每一组TLV紧密排列，中间无空格等分隔符

第二行: 需要匹配的tag数量n (0 < n <1000) 。

后面n行: 需要匹配的n个tag值（十进制表示)，递增排列。

## 输出描述

和需要匹配的n个tag对应的n行匹配结果，每一行由长度和偏移量组成

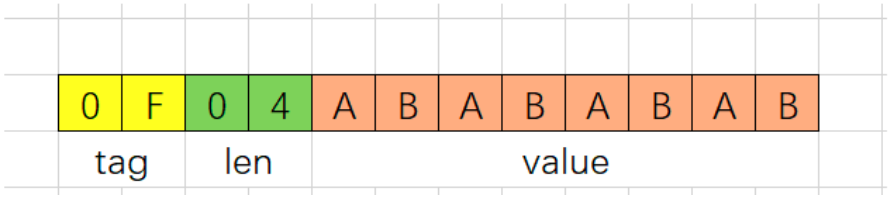
## 用例

| | |
|---|---|
| 输入 | 0F04ABABABAB<br>1<br>15 |
| 输出 | 4 2 |
| 说明 | tag15(十六进制0F)对应数据的长度为4，其value从第三个字节开始，因此偏移量为2 |

| | |
|---|---|
| 输入 | 0F04ABABABAB1001FF<br>2<br>15<br>17 |
| 输出 | 4 2<br>0 0 |
| 说明 | 第二个tag匹配失败 |

## 题目解析

以用例1为例：



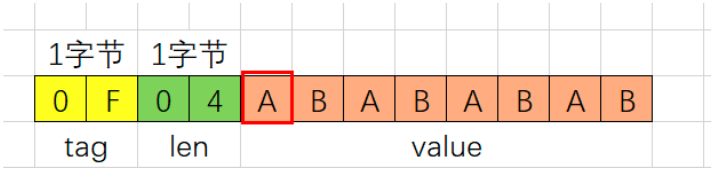

因为根据题目意思， tag,length各占1字节，1个字节 = 8 位2进制 = 2位16进制，因此：

黄色部分两位代表tag，绿色部分两位代表len，而橙色部分value的长度取决于绿色部分len的16进制值，比如用例的len= 0x04 = 4，因此value的就是4个字节，即8位16进制。

这样的话，我们就确定了输入字符串的tag,len,value。

但是，本题需要输出[len, valueOffset]

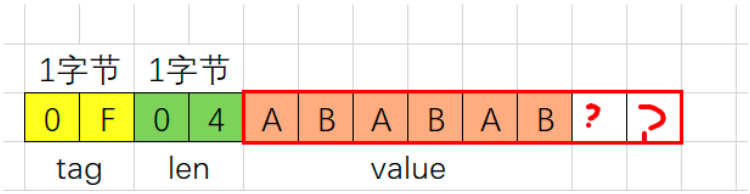
其中valueOffset，是value起始位置的在整个字符串中的（字节单位）索引，比如




用例1中的valueOffset就是2。

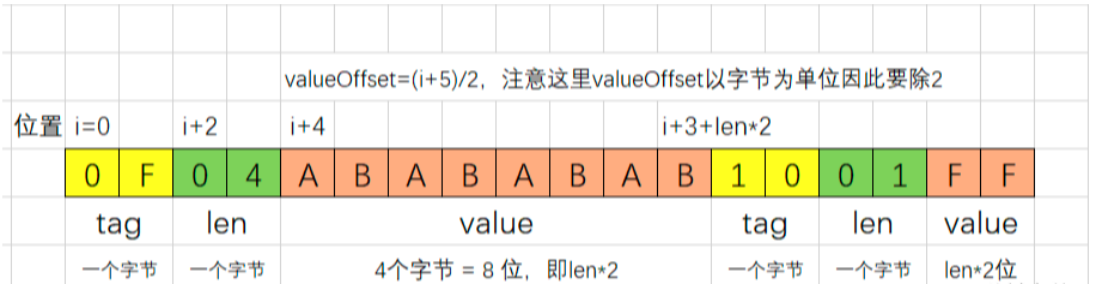
另外，本题中，有可能给的报文不是完全的，比如输入字符串为：

```
0F04ABABAB
```




按照len值，value理论上应该有4字节长度，但是实际只有3字节，因此此时报文不完整，可以丢弃。

关于下面源码中各行代码的解释




valueOffset=(i+5)/2，注意这里valueOffset以字节为单位因此要除2
位置 i=0
i+2
i+4
i+3+len*2
0 F 0 4 A B A B A B A B 1 0 0 1 F F
tag
len
value
tag
len
value
一个字节
一个字节
4个字节 = 8 位，即len*2
一个字节
一个字节
len*2位
for (let i = 0; i < msg.length; i++) {
const tag = parseInt(msg.slice(i, i + 2), 16);
const len = parseInt(msg.slice(i + 2, i + 4), 16);
const valueOffset = Math.floor((i + 5) / 2);
i = i + 3 + len * 2;
此处i指针只是移动到了当前报文的尾巴处，还需要i++,才能进入下一个报文的起始位置
if (i >= msg.length) break;
tagObj[tag] = [len, valueOffset];


valueOffset=(i+5)/2，注意这里valueOffset以字节为单位因此要除2
位置 i=0
i+2
i+4
msg.length
i+3+len*2
理论结束位置
0 F 0 4 A B A B A B
tag
len
value
一个字节
一个字节
4个字节 = 8 位，即len*2
for (let i = 0; i < msg.length; i++) {
const tag = parseInt(msg.slice(i, i + 2), 16);
const len = parseInt(msg.slice(i + 2, i + 4), 16);
const valueOffset = Math.floor((i + 5) / 2);
i = i + 3 + len * 2;
这里求出报文结束位置i，其实是为了下一步判断报文是否完整
if (i >= msg.length) break;
如果理论结束位置 > msg.length，则说明报文不完整，直接结束
tagObj[tag] = [len, valueOffset];

## JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let msg;
let n;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    msg = lines[0];
    n = lines[1] - 0;
  }

  if (n && lines.length === n + 2) {
    const tags = lines.slice(2).map(Number);
    getResult(msg, tags);
    lines.length = 0;
  }
});

function getResult(msg, tags) {
  const tagObj = {};

  // 这里i+3的目的是确保tag，len的截取不会越界，如果I+3越界了，那么说明当前TLV报文段不完整，可以直接抛弃
  for (let i = 0; i + 3 < msg.length; i++) {
    const tag = parseInt(msg.slice(i, i + 2), 16);
    const len = parseInt(msg.slice(i + 2, i + 4), 16);
    const valueOffset = Math.floor((i + 5) / 2);

    // 本TLV格式报文段结束位置i
    i = i + 3 + len * 2;

    // 如果结束位置i越界，则当前TLV报文段是一个不完整的，需要丢弃
    if (i >= msg.length) break;

    // 题目已经保证tag不会重复
    tagObj[tag] = [len, valueOffset];
  }

  tags.forEach((tag) => {
    if (tagObj[tag]) {
      const [len, valueOffset] = tagObj[tag];
      console.log(`${len} ${valueOffset}`);
    } else {
      console.log("0 0");
    }
  });
}
```

Java算法源码

```java
import java.util.HashMap;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    String msg = sc.next();

    int n = sc.nextInt();

    int[] tags = new int[n];
    for (int i = 0; i < n; i++) {
      tags[i] = sc.nextInt();
    }

    getResult(msg, tags);
  }

  public static void getResult(String msg, int[] tags) {
    HashMap<Integer, Integer[]> tagMap = new HashMap<>();

    // 这里i+3的目的是确保tag，len的截取不会越界
    for (int i = 0; i + 3 < msg.length(); i++) {
      int tag = Integer.parseInt(msg.substring(i, i + 2), 16);
      int len = Integer.parseInt(msg.substring(i + 2, i + 4), 16);
      int valueOffset = (i + 5) / 2;

      // 本TLV格式报文段结束位置i
      i += 3 + len * 2;

      // 如果结束位置i越界，则当前TLV报文段是一个不完整的，需要丢弃
      if (i >= msg.length()) break;

      // 题目已经保证tag不会重复
      tagMap.put(tag, new Integer[] {len, valueOffset});
    }

    for (int tag : tags) {
      if (tagMap.containsKey(tag)) {
        Integer[] tmp = tagMap.get(tag);
        int len = tmp[0];
        int valueOffset = tmp[1];
        System.out.println(len + " " + valueOffset);
      } else {
        System.out.println("0 0");
      }
    }
  }
}
```

## Python算法源码

```python
# 输入获取
msg = input()
n = int(input())
tags = [int(input()) for i in range(n)]


# 算法入口
def getResul(msg, tags):
    tagDict = {}

    # 这里只遍历到len(msg) - 3的目的是确保tag，len的截取不会越界
    for i in range(len(msg) - 3):
        tag = int(msg[i:i + 2], 16)
        long = int(msg[i + 2:i + 4], 16)
        valueOffset = (i + 5) // 2

        # 本TLV格式报文段结束位置i
        i += 3 + long * 2

        # 如果结束位置i越界，则当前TLV报文段是一个不完整的，需要丢弃
        if i >= len(msg):
            break

        # 题目已经保证tag不会重复
        tagDict[tag] = [long, valueOffset]

    for tag in tags:
        if tagDict.get(tag) is None:
            print("0 0")
        else:
            print(" ".join(map(str, tagDict[tag])))


# 算法调用
getResul(msg, tags)
```