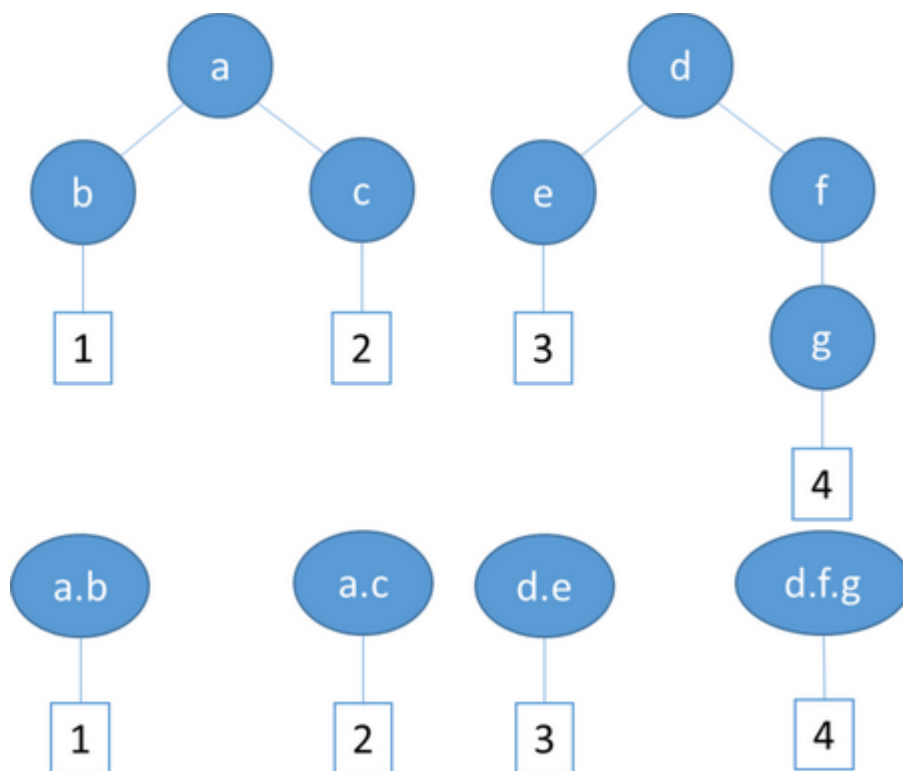


# 把一个字典扁平化

源字典 {'a':{'b':1,'c':2}, 'd':{'e':3,'f':{'g':4}}} 目标字典 {'a.c': 2, 'd.e': 3, 'd.f.g': 4, 'a.b': 1}



```
source = {'a':{'b':1,'c':2}, 'd':{'e':3,'f':{'g':4}}}
target = {}
# recursion
def flatmap(src, prefix=''):
    for k,v in src.items():
        if isinstance(v, dict):
            flatmap(v, prefix=prefix+k+'.') # 递归调用
        else:
            target[prefix+k] = v
    return target

flatmap(source)
print(target)
```

一般这种函数都会生成一个新的字典，因此改造一下 dest字典可以由内部创建，也可以有外部提供

```
source = {'a':{'b':1,'c':2}, 'd':{'e':3,'f':{'g':4}}}

# recursion
def flatmap(src, dest=None, prefix=''):
    if dest == None:
        dest = {}
    for k,v in src.items():
```

```

        if isinstance(v, dict):
            flatmap(v, dest, prefix=prefix+k+'.') # 递归调用
        else:
            dest[prefix+k] = v
    return dest

print(flatmap(source))

```

能否不暴露给外界内部的字典呢？能否函数就提供一个参数源字典，返回一个新的扁平化字典呢？递归的时候要把目标字典的引用传递多层，怎么处理？

```

source = {'a':{'b':1,'c':2}, 'd':{'e':3,'f':{'g':4}}}

# recursion
def flatmap(src, dest=None):
    if dest is None:
        dest = {}

    def _flatmap(src, prefix=""):
        for k,v in src.items():
            if isinstance(v, dict):
                _flatmap(v, prefix=prefix + k + '.')
            else:
                dest[prefix + k] = v
        #return dest

    _flatmap(src)
    return dest

print(flatmap(source))

```

## 实现Base64编码

要求自己实现算法，不用库

索引	对应字符	索引	对应字符	索引	对应字符	索引	对应字符
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w		
15	P	32	g	49	x		
16	Q	33	h	50	y		

Base64编码核心思想：每3个字节断开，拿出一个3个字节，每6个bit断开成4段。因为每个字节其实只占用了6位， $2^6 = 64$ ，因此有了base64的编码表。每一段当做一个8bit看它的值，这个值就是Base64编码表的索引值，找到对应字符。再取3个字节，同样处理，直到最后。

举例：abc对应的ASCII码为：0x61 0x62 0x63 **01100001 01100010 01100011** # abc **011000 010110 001001 100011** 00011000 00010110 00001001 00100011 # 每6位补齐为8位 24 22 9 35

问题：一个字节能变成几个Base64的字节？两个字节能变成几个Base64的字节？字符串`反引号`如何处理？

末尾的处理？1、正好3个字节，处理方式同上 2、剩1个字节或2个字节，用0补满3个字节 3、补0的字节用=表示

```
# 自己实现对一段字符串进行base64编码
alphabet = b"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
print(alphabet)

src = 'abcd'

def base64encode(src:str):
    ret = bytearray()
    if isinstance(src, str):
        _src = src.encode()
    else:
        return

    length = len(_src)
```

```

for offset in range(0, length, 3):
    triple = _src[offset:offset+3] # 切片可以超界
    #print(triple)

    r = 3 - len(triple)
    if r:
        triple += b'\x00' * r # 便于计算先补零
    #print(triple, r)

    # bytes和 bytearray都是按照字节操作的，如何按照位操作呢？需要整数
    # 将3个字节看成一个整体转成字节bytes，大端模式
    # abc => 0x616263
    b = int.from_bytes(triple, 'big')
    #print(b, hex(b))

    # 01100001 01100010 01100011 # abc
    # 011000 010110 001001 100011 # 每6位断开
    for i in range(18, -1, -6):
        #print(hex(b), i)
        index = b >> i if i == 18 else b >> i & 0x3F
        #print(index, alphabet[index])
        ret.append(alphabet[index])

    # 替换等号
    if r:
        ret[-r:] = b'=' * r # 从索引-r到末尾使用右边的多个元素依次替换
    return bytes(ret)

import base64

strlist = ['a', '', 'ab', 'abc', 'abcd', 'ManMa', '教育a']
for x in strlist:
    print(x)
    print(base64encode(src))
    print(base64.b64encode(src.encode()))
    print()

```

## 求2个字符串的最长公共子串

最长公共子串 ( LCS , Longest Common Substring )

思考：s1 = 'abcdefg' s2 = 'defabcd'

### 方法一

可否这样思考？字符串都是连续的字符，所以才有了下面的思路。

假设s1、s2两个字符串，s1短一些。

思路一：第一轮从s1中依次取1个字符，在s2查找，看是否能够找到子串。如果没有一个字符在s2中找到，说明就没有公共子串，直接退出。如果找到了至少一个公共子串，则很有可能还有更长的公共子串，可以进入下一轮。

第二轮 然后从s1中取连续的2个字符，在s2中查找，看看能否找到公共的子串。如果没找到，说明最大公共子串就是上一轮的随便的哪一个就行了。如果找到至少一个，则说明公共子串可能还可以再长一些。可以进入下一轮。

改进 其实只要找到第一轮의公共子串的索引，最长公共子串也是从它开始的，所以以后的轮次都从这些索引位置开始，可以减少比较的次数。

思路二：既然是求最大子串，最长子串不会超过最短的字符串，先把s1全长作为子串。在s2中搜索，是否返回正常的index，正常就找到了最长子串。

没有找到，把s1按照length-1取多个子串。在s2中搜索，是否能返回正常的index。

注意：不要一次把s1的所有子串生成，用不了，也不要从最短开始，因为题目要最长的。但是也要注意，万一他们的公共子串就只有一个字符，或者很少字符的，思路一就会占优势。

```
s1 = 'abcdefg'
s2 = 'defabcdoabcdeftw'
s3 = '1234a'

def findit(str1, str2):
    count = 0
    if len(str2) < len(str1):
        str1, str2 = str2, str1 # str1更短
    print(str1, str2)

    length = len(str1)
    for sublen in range(length, 0, -1):
        for start in range(0, length - sublen + 1):
            substr = str1[start:start + sublen]
            #print(substr)
            count += 1

            if str2.find(substr) > -1:
                print("count={}, substrlen={}".format(count, sublen))
                return substr

    print(findit(s1, s2))
    print(findit(s1, s3))
```

方法一，虽然做了些优化，但是还是做了很多无用功。KMP算法是这类方法中最好的实现。

## 方法二

动态规划算法

s1 = 'abcdefg' s2 = 'defabcd' 让s2的每一个元素，去分别和s1的每一个元素比较，相同就是1，不同就是0，有下面的矩阵

	s1
s1	0001000
s1	0000100
s1	0000010
s1	1000000
s1	0100000
s1	0010000
s1	0001000

上面都是s1的索引。看与斜对角线平行的线，这个线是穿过1的，那么最长的就是最长子串。 `print(s1[3:3+3])`  
`print(s1[0:0+4])` 最长

得到上面的表，需要一个字符扫描最长子串的过程，扫描的过程就是 `len(s1) * len(s2)` 次，`O(n * m)`，这是必须的。难道还需要遍历一遍找出谁才是最长的吗？能够在求矩阵过程中就找出最长的子串吗？0001000 第一行，索引为(3, 0)。第二行的时候如果索引(4, 1)处是1，就判断(3, 0)处是否为1，为1就把(3, 0)处加1。第二行的时候如果索引(5, 2)处是1，就判断(4, 1)处是否为1，是1就加1，再就判断(3, 0)处是否为1，为1就把(3, 0)加1。

	s1
s1	0003000
s1	0000200
s1	0000010
s1	4000000
s1	0300000
s1	0020000
s1	0001000

上面的方法是个递归问题，不好。最后在矩阵中找到最大的元素，从它开始就能写出最长的子串了。但是这个不好算，因为是逆推的，改为顺推。

	s1
s1	0001000
s1	0000200
s1	0000030
s1	1000000
s1	0200000
s1	0030000
s1	0004000

顺推的意思，就是如果找到一个就看前一个的数字是几，然后在它的基础上加1。

```
# www.magedu.com

def findit(str1, str2):
    length1 = len(str1)
    length2 = len(str2)
    # 行取决于str2，列取决于str1的元素个数
    matrix = [[0]*length1 for i in range(length2)]

    # 从x轴或者y轴取都可以，选择x轴，xmax和xindex
    xmax = 0
    xindex = 0
    for i, x in enumerate(str2):
        for j, y in enumerate(str1):
            if x != y: # 字符不等，什么都不做
                pass
            else: # 两字符相等
                if i == 0 or j == 0: # 在边上
                    matrix[i][j] = 1
                else:
                    matrix[i][j] = matrix[i-1][j-1] + 1

            # 记录最大值
            if matrix[i][j] > xmax:
                xmax = matrix[i][j] # 记录最大值，用于下次比较
                xindex = j

    start = xindex + 1 - xmax
    end = xindex + 1
    #print(matrix, xmax, xindex, start, end)
    return str1[start:end]

s1 = 'abcdefg'
s2 = 'defabcd'
#s2 = 'defabcdoabcdeftw'
```

```
s3 = '1234a'
s4 = "5678"
s5 = 'abcdd'

print(findit(s1, s2))
print(findit(s1, s3)) # a
print(findit(s1, s4)) # 空串
print(findit(s1, s5)) # abcd
s1 = ' abcdefg '
s5 = '304abcdd'
print(findit(s1, s5)) # abcd
```

动态规划算法，空间换时间，基本思想就是用一张表记录了所有已解决子问题的答案。