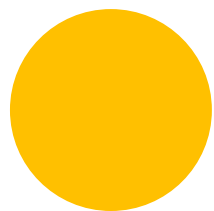
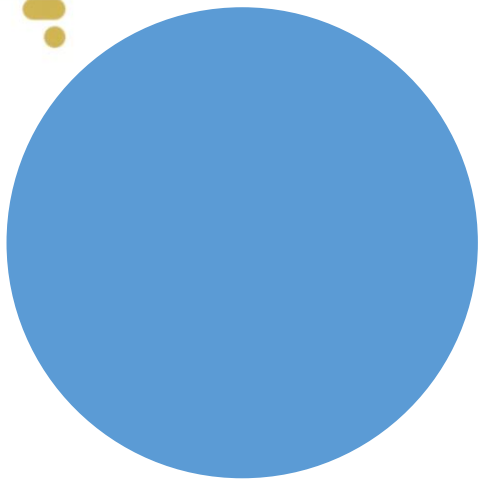




# 马上开始

35tang-C++竞赛系列三阶课程




# 《 35tang-C++竞赛系列三阶课程 》



# 本节目标

- 深化学习动态规划，学习动态规划的数组压缩优化
- 尝试去总结dp的递推公式，如何设置dp数组的含义



# DP的一个重要分析方法

---

- 求第 $n$ 项，或者给定长度是 $n$ 。那么就从长度 $1, 2, \dots, i$ 这样一段一段来看，看看知道了 $i$ 怎么得到 $i+1$ 。反过来就是知道了 $i-1$ ，怎么得到 $i$

# 最长公共子序列长度（子串）

给出长度大于1的两个序列P1 和 P2，求它们的最长公共子序列长度（不一定连续）

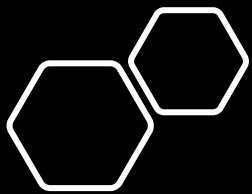
这里设P1, P2是两个字符串，数组一样操作

zabcdefg

czykdgaz

上面两个串中cdg是最长的公共子序列

DP的思考方法：这两个串的公共子序列和比他们短一些的串的公共子序列有什么关系？现在是两个串，显然和这两个串短一点都有关系

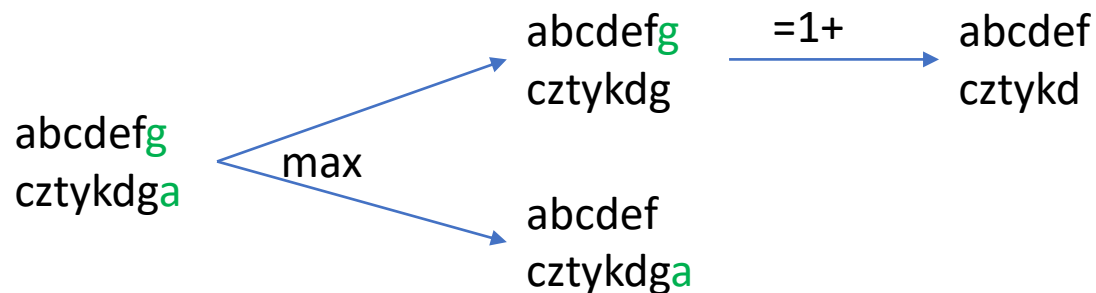


# DP

设 $dp[i][j]$ 为从第一个字符  
(数组元素下标0) 开始长  
度为 $i$ 的 $str1$ 和长度为 $j$ 的 $str2$   
的公共子子序列长度

注意：字符串字符位置从0  
开始，所以长度为 $i$ 对应的  
字符位置是 $i-1$

```
int dp[100][100]={0};
for (int i = 1; i <= m; ++i)
{
    for (int j = 1; j <= n; ++j)
    {
        if (s1[i-1] == s2[j-1])
            dp[i][j] = dp[i - 1][j - 1] + 1;
        else
            dp[i][j] = max(dp[i][j - 1], dp[i - 1][j]);
    }
}
return dp[m][n];
```



二维dp都可以反过来用表格来分析

如果结束两个字符相同，值就是左上  $(i-1, j-1)$  的值+1，不一样，就是上  $(i-1)$  和左的最大  $(j-1)$

	j	1	2	3	4	5	6	7
i		a	b	c	d	e	f	g
1	c	0	0	1	1	1	1	1
2	z	0	0	1	1	1	1	1
3	t	0	0	1	1	1	1	1
4	y	0	0	1	1	1	1	1
5	k	0	0	1	1	1	1	1
6	d	0	0	1	2	2	2	2
7	g	0	0	1	2	2	2	3
8	a	0	0	1	2	2	2	3

abcdefg  
czykdga

# 最长公共子串长度

给出长度大于1的两个字符串，求它们的最长公共子串长度（一定是连续的）。比如下面2个串最长公共子串是ab

rcogab dehoabf

一样，分析一下，两个串的最长公共子串长度和短一点的串有什么关系？现在是两个串，显然和这两个串短一点都有关系



# 如何设置dp?

如果我们还是类似设置设 $dp[i][j]$ 为从第一个字符（数组元素下标0）开始长度为i的str1和长度为j的str2的最大子串长度

rcogabe dehoabfe

发现比较最后两个字符，就算他们相等，也无法增加最大公共子串长度，比如我们最后一个字符e相等，剩下的短一点的两个串的最大公共子串长度是2，但是这个时候不能 $2+1=3$ ，因为前面短一点的串的最大公共子串（ab）并不是以最后一个字符为结束的，所以修改dp的定义：设 $dp[i][j]$ 为从第一个字符（数组元素下标0）开始长度为i的str1和长度为j的str2的最大子串长度，**该公共子串分别以i和j为结束位置。**

rcogabe dehoabfe    dp为1+ rcogab dehoabf 这两个子串的dp，因为最后一个e相同

rcogab dehoabf    最后一个字符不同，所以dp是0

rcogab dehoab    最后一个相同，多以dp是1+两个串都去掉最后一个dp

# dp公式

```
int dp[100][100]={0};  
if (str1[i-1 ] == str2[j -1])  
    dp[i][j] = 1+dp[i - 1][j - 1];  
//如果不相等， dp[i][j]还是初始化的0
```

由于dp[i][j]的定义是严格要求以第i和第j个元素结束的最大公共子串，所以最后需要打擂台法取出所有的dp里面最大的（注意，这是和前面一个例题的最大不同），虽然做了两次双循环，但是时间复杂度还是 $O(n^2)$

```
if (dp[i][j]>maxr) maxr=dp[i][j];
```

# 求一个整数数列里面的最长上升子序列长度LIS（不一定连续）

✍ 比如给定 $a=\{2,7,1,5,2,6,4,3,8,9,10,11,8\}$ ；最长上升子序列长度是7，序列为2,5,6,8,9,10,11

■ 给定{2} 显然结果就是1，就选这个2, 包含2的子序列长度是1

✎ 如果给定{2,7}， $7>2$ ，所以最长是2，序列是2，7, 包含7的子序列长度是1

■ 给定{2,7,1},  $1<7$ , 并且  $1<2$ ，所以最长还是2，依旧是2,7，包含1的子序列长度是1

■ 给定{2,7,1,5}， $5>1$  最长的可以是1,5 也可以是2，5 一样都是2, 包含5的子序列长度是2

■ 给定{2,7,1,5,2}，2和前面的去比较，新的包含2的最长子序列长度是2

■ 给定{2,7,1,5,2,6}，6和前面比较，可以和2，5组成2，5，6，也可以和1，2组成1，2，6，显然包含6的最长子序列长度是3.

■ 给定{2,7,1,5,2,6,8}，显然8可以和前面任意一个数的原有最大序列上面+1，由于6结尾的序列最长，所以现在长度是 $3+1=4$

■ 也可以倒过来看，我们找最后一个数能够参与组成的最大序列，一定是他前面所有比他小的数组成的最大序列+1。

---

# DP

---

- 定义的 $dp[i]$ 表示当前 $a[i]$ 结尾的最大上升子序列长度，那么
- $dp[0]=0$
- $dp[i]=\max(dp[j]+1) \{j<i \ \&\& \ a[j]<a[i]\}$

```
int LISfun(int a[],int n)
{
    int dp[n];dp[0]=1;
    int maxlength=1;
    for (int i=1;i<n;i++)    {
        dp[i]=1;//至少是1
        for (int j=0;j<i;j++){
            if (a[j]<a[i]) dp[i]=max(dp[i],dp[j]+1);
        }
        if (dp[i]>maxlength)    maxlength=dp[i];
    }
    return maxlength;
}

int main()
{
    int n=13;
    int a[13]={2,7,1,5,2,6,4,3,8,9,10,11,8};
    int slength=LISfun(a,n);
    cout<<slength<<endl;
    return 0;
}
```

# USACO training

## Section 2.3 PROB

### The Longest Prefix



- 如果一个集合  $P$  中的元素可以串起来（元素可以重复使用）组成一个序列  $s$ ，那么我们认为序列  $s$  可以分解为  $P$  中的元素。元素不一定要全部出现（如下例中 BBC 就没有出现）。举个例子，序列  $s$  ABABACABAAB 可以分解为下面集合中  $P$  的元素：{A, AB, BA, CA, BBC}
- 序列  $s$  的前面  $k$  个字符称作  $s$  中长度为  $k$  的前缀。设计一个程序，输入一个元素集合以及一个大写字母序列，设  $s'$  是序列  $s$  的最长前缀，使其可以分解为给出的集合  $P$  中的元素，求  $s'$  的长度  $k$ 。
- 题目比较绕，简单来说就是求一个大字符串前面最长多少个字符是由给定的一些小字符串拼成的（或者说可以分解成这些小字符串的组合）。

## 搜索--》记忆化搜索- -》 dp

---

我们先用搜索的思路来看，比如长度11的ABABACABAAB，P里面包含{A,AB,BA,CA,BBC}

---

如果要知道长度是i（方便起见，字符串从1开始，i个位置就是第i个字符）的前缀是否满足条件,先看看从i位置往前是否找得到P满足条件，如果i现在是9，串是ABABACABA，我们发现P中的BA，A都满足条件，那么我们就分别取看看截取掉BA和A的串是否满足条件，也就是ABABACA（长度7）和ABABACAB（长度8），依次递归下去就可以了。

---

用记忆化搜索的知识，我们发现比如找长度是i的某一个是否合法，可能重复在尝试，比如我们从i+1，i+2，i+3的位置都有可能找回去看i的位置。所以我们需要把S长度是i的是不是满足条件记下来，比如用dp数组。

---

定义dp[i]为长度为i是否可行，是否可行取决于dp[i-pstr[j].length()]是否可行（pstr[j]需要匹配长度为i的s串的最后）

# 倒过来递推

- 比如长度11的ABABACABAAB, P里面包含{A,AB,BA,CA,BBC}
- ABABACABAAB
- 长度为1, A可以,  $dp[1]=true$
- 长度为2, AB可以,  $dp[2]=true$
- 长度为3, ABA其中最后的BA可以匹配到, 剩下长度1, 由于 $dp[1]=true$ , 所以 $dp[3]=true$
- 长度为4, ABAB其中最后的AB可以匹配到, 剩下长度2, 由于 $dp[2]=true$ , 所以 $dp[4]=true$
- 长度为5, ABABA其中最后的BA可以匹配到, 剩下长度3, 由于 $dp[3]=true$ , 所以 $dp[5]=true$
- 长度为6, ABABAC其中最后的都匹配不上, 所以 $dp[6]=false$
- 长度为7, ABABACA其中最后的CA可以匹配到, 剩下长度5, 由于 $dp[5]=true$ , 所以 $dp[7]=true$
- 长度为8, ABABACAB其中只有最后的AB可以匹配到, 剩下长度6, 由于 $dp[6]=false$ , 所以 $dp[8]=false$
- ...



找每一个pstr里面的子串，看是否和sstr的前i个字符的最后面匹配并且除了这个子串，sstr前面的也都能够匹配，就说明长度i是可以的。



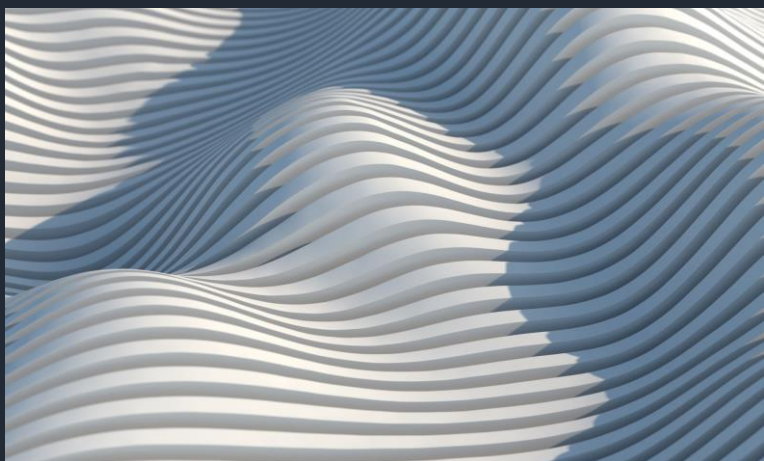
```
for ( int i=1;i<=sstr.length();i++){
    for(int j=0;j<plen;j++) {
        if ((i>pstr[j].length() &&dp[i-pstr[j].length()]!=0) || i==pstr[j].length())
        {
            if (sstr.substr(i-pstr[j].length()+1,pstr[j].length())==pstr[j]){
                //只要找到了，说明这个长度可以(dp[i]=1),
                //就没有必要去尝试pstr其他的子串了
                dp[i]=1;
                maxdp=i;
                break;
            }
        }
    }
}
```

由于i从小到大依次判断dp[i]是否可行，可行赋值给maxdp，所以最后maxdp就是答案

# USACO training

## Section 2.2 PROB

### Subset Sums



给定1-39的连续数字，把他们分成两个部分，两部分和相同，多少分法。

例如1到4的{1, 2, 3, 4}只能分成{1,4}和{2, 3}

现在问{1,2,.....39}多少分法。

---

用搜索的思路分析一下，比如a数组，{1, 2, 3..7}，首先我们需要的答案实际上是取出几个元素可以组成和为 $n*(n+1)/4$ ，也就是14，我们可以把两种情况加起来：如果选7，再看看剩下的里面取出和为7的有多少种；如果不选7，剩下的数能不能凑出14。递归下去。这样可以写出记忆化搜索：两个参数，当前要凑的和，当前从1到几里面取选

---

这样动态规划的思路就来了，设 $dp[i][j]$ 是取出和为 $i$ ，从前 $j$ 个元素里面取的分法，那么就像背包一样（下节课讲），每个元素 $j$ 都有取和不取两种可能，取得话就有 $dp[i-j][j-1]$ 种，不取的话就是 $dp[i][j-1]$ 种，两个应该加起来就是 $dp[i][j]$ 。

```
setsum=N*(N+1)/4;
```

```
for (int num=1;num<=N;num++) dp[num][num]=1;
```

```
//注意上面的初始化
```

```
for (int sum=1;sum<=setsum;sum++)
```

```
for (int num=1;num<=N;num++)
```

```
{
```

```
    dp[sum][num]+=dp[sum][num-1];
```

```
    if (sum>=num) dp[sum][num]+=dp[sum-num][num-1];
```

```
}
```

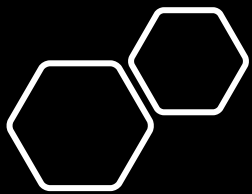
```
cout <<dp[setsum][N]/2<< endl;
```

//这里除2是因为重复了，比如dp[5][4]是在1到4选择和是5的，这里我们会选择出 (1, 4), (2, 3) 两种，实际上答案问的其实是多少组合，也就是多少对，这显然是一对。

```
-----  
Process exited after 5.549 seconds with return value 3221226356  
Press any key to continue . . .
```

tips

- 有时候程序返回如图信息：
- 数组越界？ 内存溢出？



# 作业1: 最长公共子 序列

给出两个数字序列，求其中最大的公共子序列。

输入格式：

第一行2个整数m，n（都是小于1000的正整数）

第二行依次为第一个序列的m个整数，空格分隔。

第三行依次为第二个序列的n个整数，空格分隔。

输出格式：

一行，按照顺序依次输出最大公共子序列的整数，空格分开。

示例输入：

5 5

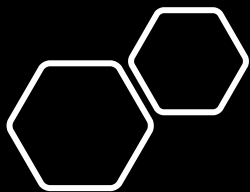
2 4 1 7 19

8 2 3 1 7

示例输出

3

示例说明：最长的公共子序列是2 1 7，长度为3



# 挑战作业2

## 数字编码

包含大写字母A到Z的字符串可以使用以下映射关系编码为数字：

'A' -> "1"

'B' -> "2"

...

'Z' -> "26"

如果给定了编码后的数字，对应的字符串可能不止一个。例如给定数字12，可能是字符串"AB"的编码也可能是字符串"L"的编码。

现在给定一个仅包含数字的非空字符串（长度小于100并且大于0），返回可能的编码为这个字符串的大写字母字符串的个数。

示例输入：

12

示例输出

2

示例输入2：

06

示例输出2

0

示例输入3：

12345671234567

示例输出3：

9

## 作业2提示

- 输入的数字字符串，如果设 $dp[i]$ 为长度为前 $i$ 个字符所对应的方案数，那么根据第 $i$ 个字符和第 $i-1$ 个字符， $dp[i]$ 和 $dp[i-1]$ 以及 $dp[i-2]$ 有什么关系？

由易到难，思维体系训练  
实战结合，创新协作培养  
兴趣导向，未来职业引领

<https://www.35tang.com>



扫码关注公众号

<https://www.三五堂.com>



添加辅导老师