# 最大公共子序列问题

公共子序列和公共子串是有区别滴，子串必须是连续的，but子序列是不必的，如ABCDDAB && BDCABA 最长公共子串只能是AB或者BD, 但是最长公共子序列是BCAB;

## 两字符串最长子序列

我们先用c[i][j]表示“a0,a1,...,a(i-1)”and squence "b0,b1,...,b(j-1)"的最长子序列长度，最后再逆序访问数组C[i][j]来输出最长公共子序列

例如对于样例a[]="12345" && b[]=“145 ”，开辟数组空间c[5+1][3+1], c[i][j]有：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 真实值 |  |  | 1 | 4 | 5 |
|  | 序号 | 0 | **1** | **2** | **3** |
|  | 0 | 0 | 0 | 0 | 0 |
| 1 | **1** | 0 | 1 | 1 | 1 |
| 2 | **2** | 0 | 1 | 1 | 1 |
| 3 | **3** | 0 | 1 A | 1 | 1 B |
| 4 | **4** | 0 | 1 | 2 C | 2 D |
| 5 | **5** | 0 | 1 | 2 | 3 |

好高深的算法：时间复杂度为m\*n；我们来看位置C，是在比较1234与14的公共子序列，if(4和4相同)，所以位置A+1；

接着看位置D，是比较1234与145，4和5不相等，那么就在C和B中选一个大的值吧

普世做法：

1. 你的背包，迪杰斯特拉最短路径等，先顺序求出长度数组，这个时候就要多开辟一个空间，边界值0初始化为0，然后比较c[i] and c[i-1], 字符串从1开始编号了
2. 逆序访问这个长度数组，求出真正的序列；

或者每次c[i]==c[j]时，记录下当前字符

1. 当然导弹拦截这种，是直接逆序求长度数组的，就不用同步骤1一样多开辟一个空间了，

### 代码C++版本

//2016-5-12

#include<iostream>

using namespace std;

char a[100] = "12345678906", b[100] = "44445666";

//char a[100] = "12345", b[100] = "145";

int c[100][100];

//c[i][j]表示“a0,a1,...,a(i-1)”and squence "b0,b1,...,b(j-1)"的最长子序列长度

int lcsLen(char a[], char b[], int c[][100]){

int m = strlen(a), n = strlen(b);

for(int i=0; i<=m; i++)

c[i][0] = 0;

for(int i=0; i<=n; i++)

c[0][i] = 0;

for(int i=1; i<=m; i++){

for(int j=1; j<=n; j++){

if(a[i-1] == b[j-1])

c[i][j] = c[i-1][j-1] + 1;

else if(c[i-1][j] >= c[i][j-1])

c[i][j] = c[i-1][j];

else

c[i][j] = c[i][j-1];

}

}

cout<<"let's see see C[i][j]"<<endl;

for(int i=0; i<=m; i++){

for(int j=0; j<=n; j++)

cout<<c[i][j]<<" ";

cout<<endl;

}

return c[m][n];

}

char\* buildLCS(char s[]){

int i = strlen(a), j = strlen(b);

//k只是得到了最长公共子序列长度，

int k = lcsLen(a, b, c);

s[k] = '\0';

//接下来得像01背包那样逆序输出序列了

while(k > 0){

if(c[i][j] == c[i-1][j])

i--;

else if(c[i][j] == c[i-1][j-1])

j--;

else{

s[--k] = a[i-1];

//s[--k] = b[j-1];

i--; j--;

}

}

return s;

}

int main(){

char str[100];

cout<<"LCS = "<<buildLCS(str)<<endl;

system("pause");

return 1;

}

### 代码java版

/\*\*

\* 求解最长公共子序列哦，很强大和牛；这个复杂度是m\*n

\* **@author** junshu

\* **@date**: 20160522

\*/

**public** **class** CommonSequence {

// String first = "12345"; String second = "145";

String first = "12345678906", second = "44445666";

**int** c[][];

**void** buildSequence(){

**char**[] firChars = first.toCharArray();

**char**[] secChars = second.toCharArray();

**int** firSize = firChars.length;

**int** secSize = secChars.length;

c = **new** **int**[firSize+1][secSize+1];

**for**(**int** i=1; i<=first.length(); i++){

**for**(**int** j=1; j<=second.length(); j++){

**if**(firChars[i-1] == secChars[j-1])

c[i][j] = c[i-1][j-1] + 1;

**else** **if**(c[i][j-1] > c[i-1][j])

c[i][j] = c[i][j-1];

**else**

c[i][j] = c[i-1][j];

}

}

**int** k = c[firSize][secSize];

System.*out*.println("the max length: " + k);

**int** i = firSize, j = secSize;

**char**[] s = **new** **char**[k];

**while**(k > 0){

**if**(c[i][j] == c[i-1][j])

i--;

**else** **if**(c[i][j] == c[i-1][j-1])

j--;

**else**{

s[--k] = firChars[i-1];

//s[--k] = secChars[j-1];

i--; j--;

}

}

System.*out*.print("The last result: ");

**for**(**char** ch: s)

System.*out*.print(ch);

System.*out*.println("\nGame Over!");

}

**public** **static** **void** main(String[] args) {

CommonSequence cs = **new** CommonSequence();

cs.buildSequence();

}

}

### 代码python版本

*# Author: shujun  
# Date: 2020-02-16 17:01  
# Place: 南昌西-上海，G1388高铁  
  
# 题目：两个字符串 "12345" && “145”，求最长公共子序列***class** Solution:  
 **def** longestSubseq(self, s1:str, s2:str):  
 subSeq = **''** *#1. 初始化二维数组记录所有情况: len(s1)+1 \* len(s2)+1  
 # 这里的行列很容易写错* matrix = [[0 **for** i **in** range(len(s2)+1)] **for** i **in** range(len(s1)+1)]  
 *# 为了防止matrix[i-1][j-1]数组越界，index从1开始* **for** i **in** range(1, len(s1)+1):  
 **for** j **in** range(1, len(s2)+1):  
 *#2.1 若字符相等，则记录长度+1* **if** s1[i-1] == s2[j-1]:  
 matrix[i][j] = matrix[i-1][j-1] + 1  
 subSeq += s1[i-1]  
 *#2.2 不相等则取相邻最大的那个* **else**:  
 **if** matrix[i][j-1] > matrix[i-1][j]:  
 matrix[i][j] = matrix[i][j-1]  
 **else**:  
 matrix[i][j] = matrix[i-1][j]  
  
 print(**"max common sequence lenth: "**, matrix[i][j])  
 **return** subSeq  
  
  
solution = Solution()  
print(solution.longestSubseq(**"12345"**, **"145"**))  
print(solution.longestSubseq(**"12345"**, **"165"**))

## 导弹拦截问题

敌人发来n枚导弹，高度为h1, h2, … , hn; 我方有导弹拦截系统，第一发任意高度，以后一发比一发低，求最多能拦截多少导弹?

实际上为求最长递减序列；

例如对于int x[] = {389, 207, 155, 300, 299, 170, 158, 65};

我们可以引入d[i](一维数组)表示如果从第i枚导弹开始拦截，d[i]表示可以拦截的数量；

假设我们从第1枚导弹高度207开始拦截，你就往后找比207高度低的导弹，

1）155是的，数量2+1=3；

2）170是的，数量3+1=4>3; 所以d[1] = 4了；

3）158是的，数量2+1=3<4; 不操作了

4）65是的，但是我不想说了

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|  | 389 | 207 | 155 | 300 | 299 | 170 | 158 | 65 |
| 初始化 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| i： 6 |  |  |  |  |  |  | 2 | 1 |
| 5 |  |  |  |  |  | 3 | 2 | 1 |
| 4 |  |  |  |  | 4 | 3 | 2 | 1 |
| 3 |  |  |  | 5 | 4 | 3 | 2 | 1 |
| 2 |  |  | 2 | 5 | 4 | 3 | 2 | 1 |
| 1 |  | 4 | 2 | 5 | 4 | 3 | 2 | 1 |
| 0 | 6 | 4 | 2 | 5 | 4 | 3 | 2 | 1 |

### C++代码：

#include<iostream>

usingnamespace std;

int x[] = {389, 207, 155, 300, 299, 170, 158, 65};

int size = sizeof(x)/sizeof(int);

int d[100];

void prior(){

for(int i=0; i<size; i++)

d[i] = 1;

//动态规划，从后面向前循环

for(int i=size-2; i>=0; i--){

//实际就是在从（j-size）中,找出比当前x[i]相差最下的那个数

for(int j=i+1; j<size; j++)

if(x[j]<x[i] && d[i]<d[j]+1)

d[i] = d[j] + 1;

}

}

void findMax(){

int dMax = 0, xh;

for(int i=0; i<size; i++){

if(d[i] > dMax){

dMax = d[i];

xh = i; //the height of the first intercepted missile

}

}

cout<<"最多可以拦截导弹数为："<<d[xh]<<endl;

cout<<"高度分别是：";

cout<<x[xh]<<" ";

for(int i=xh+1; i<size; i++){

if(x[i]<=x[xh] && d[i]==d[xh]-1){

cout<<x[i]<<" ";

xh = i;

}

}

cout<<endl;

}

int main(){

prior();

findMax();

system("pause");

return 1;

}

# Leetcode5最长回文子串

[最长回文子串 - leetcode](https://leetcode-cn.com/problems/longest-palindromic-substring)

## 动态规划之01背包

### 二维数组

Str1= abcdef; 正序

Str2 =“fedcba” 逆序

整体思想就是，申请一个二维的数组初始化为 0，然后判断对应的字符是否相等，相等的话

arr [ i ][ j ] = arr [ i - 1 ][ j - 1] + 1 。

当 i = 0 或者 j = 0 的时候单独分析，字符相等的话 arr [ i ][ j ] 就赋为 1 。

arr [ i ][ j ] 保存的就是公共子串的长度。

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  |
|  |  | a | a | c | a | b | k | a | c | a | a | 倒序 |
| **0** | a | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |  |
| **1** | a | **1** | 2 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 2 |  |
| **2** | c | 0 | 0 | 3 | 0 |  |  |  | 2 | 0 | 0 |  |
| **3** | a | 1 | 1 | 0 | 4 | 0 | 0 | 1 | 0 | 3 | 1 |  |
| **4** | k |  |  |  |  |  | 1 | 0 | 0 | 0 | 0 |  |
| **5** | b |  |  |  |  |  |  |  |  |  |  |  |
| **6** | a |  |  |  |  |  |  |  |  |  |  |  |
| **7** | c |  |  |  |  |  |  |  |  |  |  |  |
| **8** | a |  |  |  |  |  |  |  |  |  |  |  |
| **9** | a |  |  |  |  |  |  |  |  |  |  |  |
|  | 正序 |  |  |  |  |  |  |  |  |  |  |  |

### 一维数组

如果当s[i] != reversed\_s[j]时，不重写置为0，会打印出以下数组

0 [1, 1, 0, 1, 0, 0, 1, 0, 1, 1]

1 [1, 2, 0, 1, 0, 0, 1, 0, 1, 2]

2 [1, 2, 3, 1, 0, 0, 1, 2, 1, 2]

3 [1, 2, 3, 4, 0, 0, 1, 2, 3, 2]

4 [1, 2, 3, 4, 0, 1, 1, 2, 3, 2]

5 [1, 2, 3, 4, 5, 1, 1, 2, 3, 2]

6 [1, 2, 3, 4, 5, 1, 2, 2, 3, 4]

7 [1, 2, 3, 4, 5, 1, 2, 3, 3, 4]

8 [1, 2, 3, 4, 5, 1, 2, 3, 4, 4]

9 [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

[1, 2, 3, 4, 5, 1, 2, 3, 4, 5] 3 4

False

## 动态规划二

dp[i][j] = s[i] == s[j] **and** (j == i + 1 **or** dp[i+1][j-1])

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | 0 | 1 | 2 | 3 |
|  |  | a | a | c | a |
| **0** | a | T | T | F |  |
| **1** | a |  | T | F |  |
| **2** | c |  |  | T |  |
| **3** | a |  |  |  | T |