,12，动态规划整章注解

2018年4月6日 15:37

从第六题的例子可以分析出，动态规划的本质是申请额外的空间，来记录每一个暴力搜索的计算结果，下次要用时直接使用，而不用再重复的递归过程。

并且，比起记忆搜索，其定义了递归的顺序，从简单的开始递归

因此一般暴力穷举的问题都可以用动态规划来做，这个步骤一般如下

1，实现暴力递归过程

2，在暴力搜索方法的函数中看哪些参数可以代表递归过程

3，找到这些参数后，记忆化搜索的方法就很容易实现，

4，然后通过分析记忆化搜索的依赖路径，进而实现动态规划

5，根据记忆化搜索方法改出动态规划方法，进而看看是否能化简，若能，还能实现时间复杂度更低的动态规划方法。

其实动态规划的本质就是一些先贤们，在解决暴力搜索问题时，找出的以空间换时间的捷径。

总结

1,零钱问题

```
if(j - changes[i]<0)//完全不用货币changes[i]的情况
   dp[i][j] = dp[i-1][j];
else//用changes[i]货币1-n的情况
   dp[i][j] = dp[i][j-changes[i]]+dp[i-1][j];
```

2,台阶问题：dp[i] = dp[i-1]+dp[i-2];

3,矩阵最小路径和的问题，

第一行第一列累加即可，

然后的就是找左，上较小的那个加上第i，j处格点的值即可

4,最长递增子序列的问题

计算以每一个结尾的最长递增子序列长dp[i]（j从0到i-1, a[j]<a[i], 中最大的dp[j]+1)

然后求最大值

5,最长公共子序列

Dp[n+1][m+1],然后第一行第一列为0，dp[i][j]为str1[i-1],str2[i-2]的最长公共子序列长。

因此，如果str1[i-1] == str2[j-1], dp[i][j] = dp[i-1][j-1]+1;

Else dp[i][j]= max[dp[i][j-1],dp[i-1][j]);

6,背包问题

Dp[i][j]表示用0-i件物品，重量不大于j的前提下的最大价值

对于dp[i][j],如果放入了第i件物品，dp[i][j] = dp[i-1][j-w[i] + v[i];

否则，dp[i][j] = dp[i-1][j];

取两者最大即可。

1，

屏幕剪辑的捕获时间: 2018/4/6 15:38

```
/*
问题：0-1背包问题
一个背包有一定的承重M，有N件物品，每件都有自己的价值，记录在数组v中，也有自己的重量，记录
在数组w中，每件物品可以选择
装入or不装入背包，在承重范围内，选出物品的总价值最大。

解题思路：
假设物品编号为1-n，一件件放入背包
dp[n][M+1];

第1列，表示重量上限为0，那么价值即dp[i][0] = 0;
第一行，重量上限>=w[0]的，价值dp[0][i] = v[0],其余也为0.

dp[x][y]表示前x件物品，不超过重量y的时候的最大价值

此时，如果放入了第x件物品，那么前x-1件物品的重量不大于y-w[x];
如果不选，那么前x-1件的物品重量可以达到y。
因此，
dp[x][y] = dp[x-1][y-w[x]] + v[x];//放入x
dp[x][y] = dp[x-1][y];//不放入x

选两者情况中dp最大的。
*/
#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
    int solution(int m,int n,vector<int> v,vector<int> w)//承重，物品数量，价值，物品重量
    {
        int dp[n][m+1];
        //初始化第一列，即重量为0，那么价值都为0
        for(int i = 0;i<n;i++)
            dp[i][0] = 0;
        //初始化第一行，重量上限 大于等于物品0的，价值为物品0的价值
        for(int i = 0;i<=m;i++)
        {
            if(i>=w[0])
            {
                dp[0][i] = v[0];
            }
            else
                dp[0][i]= 0;
```

```
        }
        for(int i = 1;i<n;i++)
        {
            for(int j = 1;j<=m;j++)
            {
                if(j - w[i] >= 0)
                    dp[i][j] = max(dp[i-1][j-w[i]]+v[i],dp[i-1][j]);//放入第i件物品和不放
第i件物品两种情况的最大值
                else
                    dp[i][j] = dp[i-1][j];
            }
        }
        return dp[n-1][m];
    }
};



int main()
{
    int m = 297;
    int n = 8;
    int w[n] = {42,25,30,35,42,21,26,28};
    int v[n]= {261,247,419,133,391,456,374,591};
    vector<int> w_arr(w,w+n);
    vector<int> v_arr(v,v+n);
    Solution s;
    int res = s.solution(m,n,v_arr,w_arr);
    cout<<res<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

2，

屏幕剪辑的捕获时间: 2018/4/6 15:38

/*问题：最长公共子序列

给定两个字符串str1，str2,返回两个字符串的最长公共子序列，

如，str1 = “1A2C3D4B56”, str2 = "B1D23CA45B6A",最长公共子序列为123456or12C4B6,返回哪一个都行

解题思路：

如果str1长为n，str2长为m，那么定义一个dp[n+1][m+1]的数组

首先，第一行和第一列都初始化为0，然后dp[i][j]处代表的是str1[i-1],str2[j-1]处的最长公共子序列

dp[i][j]的值要怎么计算？
dp[i-1][j-1] +1 ,if str1[i] = str2[j]
max(dp[i][j-1],dp[i-1,j]) ,if !=

```
*/
#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
    int solution(string str1,string str2,int n,int m)
    {

        int dp[n+1][m+1];
        for(int i=0;i<=n;i++)//初始化第一列
            dp[i][0] = 0;
        for(int i = 0; i<=m;i++)//初始化第一行
            dp[0][i] = 0;
        for(int i = 1;i<n+1;i++)
        {
            for(int j = 1;j<m+1;j++)
            {
                if(str1[i-1] == str2[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[n][m];
    }
};


int main()
{
    string str1 = "1A2C3D4B56";
    string str2 = "B1D23CA45B6A";
    Solution s;
    int res = s.solution(str1,str2,str1.size(),str2.size());
    cout<<res<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

3，

屏幕剪辑的捕获时间: 2018/4/6 15:39

,

```
/*
问题：最长递增子序列：给定数组arr，返回arr中最长递增子序列的长度，比如
2 1 5 3 6 4 8 9 7，最长递增子序列为1 3 4 8 9，故返回5
注意这里是子序列

解题思路：定义一个dp[n],其中dp[i]代表必须以arr[i]结尾的前提下，arr[0,...,i]中最大递增子序
列长度
明显，dp[0] = 1;
其他情况，dp[i] = max(dp[j])+1,其中i>j>=0 && a[j]<a[i];

再得到dp数组中的最大值即可
*/
#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
    int solution(vector<int> arr,int n)
    {
        if(n <= 0)
            return 0;
        int *dp = new int[n];
        dp[0]= 1;
        int max_l = -1;//找最大的dp[i]
        for(int i = 1;i<n;i++)
        {
            int max_len= -1;//找j中最大的dp[j]
            for(int j = 0;j<i;j++)
            {
                if(arr[j]<arr[i])
                {
                    max_len = max(max_len,dp[j]);
                }

            }
            dp[i] = max_len+1;

            if(dp[i]>max_l)
                max_l = dp[i];

        }
        return max_l;
```

```
    }
};
int main()
{
    int n = 9;
    int a[n]={2, 1 ,5 ,3 ,6 ,4 ,8 ,9 ,7};
    vector<int> arr(a,a+n);
    Solution s;
    int res = s.solution(arr,n);
    cout<<res<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

4，

直通BAT面试算法精讲课 > 动态规划 > 12.5 矩阵最小路径和练习题

下一题

有一个矩阵map，它每个格子有一个权值。从左上角的格子开始每次只能向右或者向下走，最后到达右下角的位置，路径上所有的数字累加起来就是路径和，返回所有的路径中最小的路径和。
给定一个矩阵**map**及它的行数**n**和列数**m**，请返回最小路径和。保证行列数均小于等于100.
测试样例：

```
[[1,2,3],[1,1,1]],2,3
```

返回：4

C/C++ (clang++ 3.3)

```
1 class MinimumPath {
2 public:
3     int getMin(vector<vector<int> > map, int n, int m)
4     {
5         int **dp=new int *[n];
6         for(int i=0;i<n;i++)
7             dp[i]=new int[m];
8         for(int i=0;i<n;i++)
```

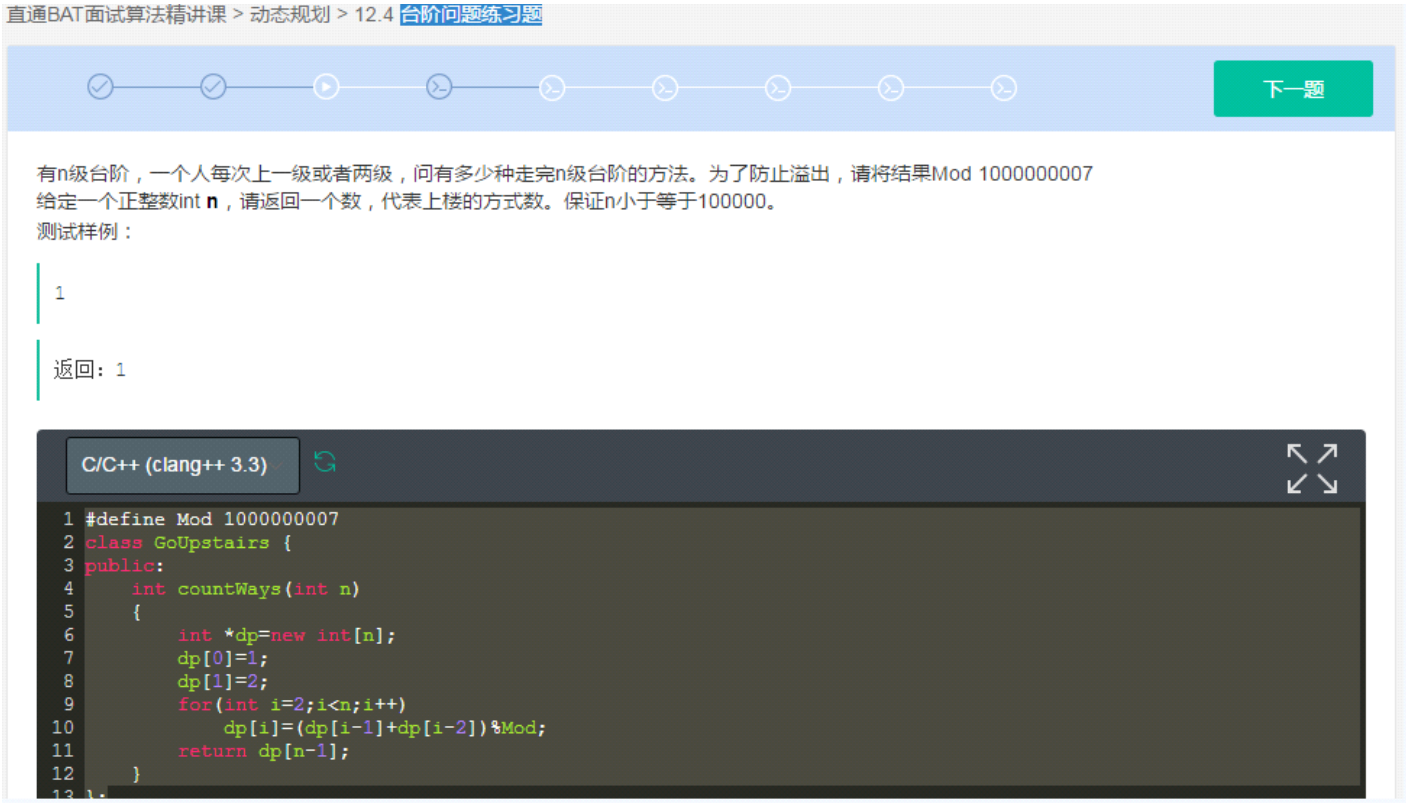
屏幕剪辑的捕获时间: 2018/4/6 15:39

```
/*
问题：给定一个矩阵m，从左上角开始每次只能向右或往下走，最后到达右下角的位置，路径上的所有
数字累加起来就是路径和，返回所有路径中最小的路径
和，如
1 3 5 9
8 1 3 4
5 0 6 1
8 8 4 0
路径1,3,1,0,6,1,0,是所有路径中最小的，所以返回12

思路：假如矩阵m的大小为M×N，那么生成一个一样大小的矩阵，dp[M][N],dp[i][j]的值表示从左上角
0,0,走到i,j
位置的最小路径和（包含位置i，j）

首先dp矩阵的第一行，由于矩阵m每次只能往下or往右走，因此，第一行上dp值为m矩阵第一行不断累加
的结果，同理
第一列上的也是。

而位置dp[i][j]处的值，一定是从dp[i-1][j]（上）,or dp[i][j-1]（左）处过来的，两者中的最小
值，再加上m[i][j]此处的值。
即dp[i][j]= m[i][j]+min(dp[i-1][j],dp[i][j-1])*/

#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
    int solution(vector<vector<int> > map,int m,int n)//m行，n列
```

```
    {
        int dp[m][n];
        //初始化dp第一行第一列
        dp[0][0] = map[0][0];
        for(int i = 1;i<n;i++ )//第一行
            dp[0][i] = dp[0][i-1]+map[0][i];
        for(int i = 1;i<m;i++)//第一列
            dp[i][0] = dp[i-1][0]+map[i][0];
        for(int i = 1;i<m;i++)
        {
            for(int j = 1;j<n;j++)
            {
                dp[i][j] = map[i][j]+min(dp[i-1][j],dp[i][j-1]);
            }
        }
        return dp[m-1][n-1];
    }
};

int main()
{
    int m,n;
    cout<<"请输入矩阵行数："<<endl;
    cin>>m;
    cout<<"请输入矩阵列数："<<endl;
    cin>>n;
    vector<vector<int> > map;
    for(int i = 0;i<m;i++)
    {
        vector<int> v;
        int c;
        for(int i = 0;i<n;i++)
        {
            cin>>c;
            v.push_back(c);
        }
        map.push_back(v);
    }
    Solution s;
    long long res = s.solution(map,m,n);
    cout<<res<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

5，



屏幕剪辑的捕获时间: 2018/4/6 15:39

https://www.nowcoder.com/questionTerminal/8c82a5b80378478f9484d87d1c5f12a4
/*问题：有n阶台阶，一个人每次往上走1or2阶，问有多少种走完的方法

思路，我们分析得知，走上第i阶台阶，只能有从第i-1处上一个台阶，or从i-2处上两个台阶。

因此有 f(i) = f(i-1)+f(i-2)

且易知，f1 = 1， f2 = 2;*/

```
#include<bits/stdc++.h>
#define Mod 1000000007
using namespace std;

//这里由于方法数很大，中间的结果也要mod
class Solution
{
public:
    int solution(int n)
    {
        int *dp = new int[n];
        dp[0]=1 ;//一阶台阶
        dp[1] =2; //2jie
        for(int i = 2;i<n;i++)
            dp[i] = (dp[i-1]+dp[i-1])%Mod;
        return dp[n-1];
    }
};
int main()
{
    int n = 100;//台阶数
    Solution s;
    long long res = s.solution(n);//此外一定要注意对于结果数很大的情况，一定用longlong存贮
    cout<<res<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

6，

屏幕剪辑的捕获时间: 2018/4/6 15:39

https://www.nowcoder.com/questionTerminal/185dc37412de446bbfff6bd21e4356ec

/*问题：给定数组arr，其中所有的值都为正数且不重复，每个值代表一种货币，每种面值的货币可以使用任意张，再
给定一个整数aim代表要找的钱数，其换钱有多少种方法。

Arr = {5,10,25,1},aim = 1000

暴力枚举:49959s,142511
记忆搜索:269s,142511
动态规划:34s,142511,时间复杂度为n×aim^2，化简后，即将1-n张arr[i]的情况，合起来，合为dp[i]dp[j-arr[i]);
此时时间复杂度为N*aim

*/

```
#include<bits/stdc++.h>
using namespace std;
/*方法1：暴力搜索方法
1,用0张5元的货币，让{10,25,1}组成剩下的1000，此时所有的方法数记为res1；
2，用1张5元的货币，让{10，25,1}组成剩下的950，此时所有的的方法数记为res2；
3，..用200张5元的货币，让{10，25,1}组成剩下的0，此时方法数记为res201，

在计算每种方法数时，需要对{10,25,1}这个序列进行递归。

分析上面的递归过程，我们定义int p1（arr，index，aim）函数，它的含义是如果用
arr[index,…N-1)的面值
的前组成aim，返回总的方法数。

存在的问题：有很多重复计算的情况
*/
class BaoliExchange{
public:
    int baoli(vector<int> arr,int n,int aim)
    {
        if(n == 0|| aim<0)
            return 0;
        return propress(arr, 0, aim);
    }

    int propress(vector<int> arr,int index,int aim)
    {
        int res;
        if(index == arr.size()-1)
        {
            res = (aim%arr[index])?0:1;///现在是到了最后一种货币，如果剩下的aim不能整除该货币，则返回0种方法
            return res;
```

```
        }
        else
        {
            res = 0;//计算每一种货币的累加情况，然后将每一种货币的返回值res累加
            for(int i = 0; arr[index]*i <= aim;i++)
                res+=propress(arr,index+1,aim-i*arr[index]);
        }
        return res;
    }
};

/*方法2，记忆搜索方法，我们注意到上面的方法中，arr是一直没变的，因此我们可以将递归函数
该为p1(int index,int aim).并记录下每次计算的结果，从而避免了大量的重复计算。

即构建一个表map，每计算完一个p（index，aim），都将结果放入map，idex和aim组成共同的key
然后每次递归过程前，先在map中查找是否计算过，否则再递归
*/


class RemerberExchange
{
public:
    int remerberfind(vector<int> arr,int n,int aim)
    {
        if(n == 0 || aim<0)
            return 0;
        //定义二维动态数组
        int **map = new int*[n];
        for(int i = 0; i<n;i++)
            map[i] = new int[aim+1];
        //并初始化
        for(int i = 0;i<n;i++ )
            for(int j = 0; j< aim+1;j++)
                map[i][j] = -1;
        int res =  propress(arr,0,aim,map);
        //释放内存，需多次析构
        for(int i = 0;i<n;i++)
            delete []map[i];
        delete []map;
        return res;

    }


    int propress(vector<int> arr,int index,int aim,int **map)
    {
        int res;
        if(index == arr.size()-1)
        {
            if(map[index][aim] ==-1)
                map[index][aim] = (aim % arr[index])?0:1;
            return map[index][aim];
        }
        else
        {
            res = 0;
            for(int i = 0; arr[index]*i<=aim;i++ )
            {
                //存贮index+1，aim-i*arr[index]处的计算值。
                if(map[index+1][aim-i*arr[index]] == -1)
                    map[index+1][aim-i*arr[index]] = propress(arr,index+1,aim-
i*arr[index],map);
                res+=map[index+1][aim-i*arr[index]];
            }
        }
        return res;
    }
};
/*动态规划
生成dp[n][aim+1]矩阵，其中dp[i][j]表示在使用arr[0,i]货币的情况下，组成钱数j有多少种方法

首先初始化，第一列，第一列是指j为0的情况，即组成钱数为0的情况，明显只有一种方法，即不使用
```

任何货币
然后初始化第一行，第一行中，只有aim值为arr[0]= 5的整数倍的地方才能被组成，因此只有这些位置的方法为1，其他为0.

那么dp[i][j]的值如何求呢
1，当完全不用arr[i]货币，只用arr[0,...,i-1]时，方法数为+dp[i-1][j];

2,如果用1张，那么方法数为+dp[i-1][j-1*arr[i]]

3,如果用n张，那么方法数为+dp[i-1][j-n*arr[i]]

用1-n张arr[i]的情况，合起来，即dp[i]dp[j-arr[i]);
*/

```
class DynamicExchange
{
public:
    int DynamicFind(vector<int> arr,int n,int aim)
    {
        if(n == 0 || aim<0)
            return 0;
        int dp[n][aim+1];
        for(int i = 0; i<n;i++)//初始化第一列
            dp[i][0] = 1;
        for(int i = 0;i<aim+1;i++)//初始化第一行
        {
            if(i % arr[0] == 0)
                dp[0][i] = 1;
            else
                dp[0][i] = 0;
        }
        for(int i = 1;i<n;i++)
        {
            for(int j = 1;j<aim+1;j++)
            {
                if(j-arr[i]<0)
                    dp[i][j] = dp[i-1][j];
                else
                    dp[i][j] = dp[i][j-arr[i]]+dp[i-1][j];
            }
        }
        return dp[n-1][aim];
    }

};



int main()
{
    int n = 4;
    int a[n] = {5,10,25,1};
    vector<int> arr(a,a+n);
    int aim = 1000;

    BaoliExchange b;
    clock_t starttime = clock();
    int num = b.baoli(arr,n,aim);
    clock_t endtime = clock();
    cout<<"暴力枚举:"<<endtime-starttime<<"s,"<<num<<endl;

    RemerberExchange r;
    starttime = clock();
    num = r.remerberfind(arr,n,aim);
    endtime = clock();
    cout<<"记忆搜索:"<<endtime-starttime<<"s,"<<num<<endl;

    DynamicExchange d;
    starttime = clock();
    num = d.DynamicFind(arr,n,aim);
    endtime = clock();
    cout<<"动态规划:"<<endtime-starttime<<"s,"<<num<<endl;
```

```
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

,7 ,

屏幕剪辑的捕获时间: 2018/4/6 15:39

```
/*
问题：给定两个字符串str1和str2，再给定三个整数ic，dc和rc，分别代表插入删除和替换一个字符的代价，返回将str1编辑成str2
的最小代价。
如：str1 = abc，str2 = adc，ic = 5，dc = 3，rc = 2，那么将b替换为d，代价是最小的，为2
再如：str1 = abc，str2=adc，ic = 5，dc = 3,rc =100,那么先删除b，再插入d的代价是最小的，为8
思路：

如果str1的长度为N，str2的长度为M，定义dp[N+1][M+1]
dp[i][j]为将str1[0,...,i-1]编辑成str2[0,...,j-1]的最小代价

自然的，dp[0][0] = 0;
然后第一列，代表将str1[0,...,i-1]编辑成空串的最小代价，自然为i*dc(0-i-1是i个字符）;即dp[i][0] = i*dc
然后第一行，代表将空串编辑乘str2[0,...,j-1](0-j-1,j个字符)的代价，自然为j*ic;

然后dp[i][j]有四种情况
1，将str1[0,i-1],删除一个字符(代价：dc)，变成str1[0,i-2],再将str1[0,i-2]编辑成str2[0,j-1](代价：dp[i-1][j]),
    此时 dp[i][j] = dc + dp[i-1][j];

2,将str1[0,i-1]编辑成str2[0,j-2](代价为dp[i][j-1]),再插入一个字符，将str2[0,j-2]变成str2[0,j-1]
    此时dp[i][j] = ir+dp[i][j-1];

3,如果str1[i-1]!=str2[j-1],那么，先将str1[0,i-2]编辑成str2[0,j-2],然后再替换i-1处的值
    此时dp[i][j] = rc+dp[i-1][j-1];

4,如果str1[i-1] = str2[j-1],那么，将str1[0,i-2]编辑成str2[0,j-2]即可
    此时dp[i][j] = dp[i-1][j-1];

然后，我们选择以上四种情况下最小的值，作为dp[i][j]的值
*/

#include<bits/stdc++.h>
using namespace std;
class Solution
```

```
{
public:
    //串1，串2,串1的长，串2的长，插入代价，替换代价，删除代价
    int solution(string str1,string str2,int n,int m,int ic,int rc,int dc)
    {
        int dp[n+1][m+1];
        dp[0][0] = 0;
        for(int i = 1;i<=n;i++)//处理第一列,将str1[0,i-1]删除成空串，即dc*i
            dp[i][0] = i*dc;
        //处理第一行，即将空串插入成str2[0,i-1],即ic*i
        for(int i = 1;i<=m;i++)
            dp[0][i]= i*ic;
        //处理其他情况，四种情况取最小值
        for(int i =1 ;i<=n;i++)
        {
            int res;
            for(int j = 1;j<=n;j++)
            {
                //先取ic+dp[i][j-1],(先将str1[0,i-1]编辑成str2[i-2],再插入)
                //和dc+dp[i-1][j]，（先将str1删除字符变成str1[0,i-2],再编辑成str2[0,j-1]的
最小值
                int temp1 = min(ic+dp[i][j-1],dc+dp[i-1][j]);

                //再看str1[i-1]和str2[j-1]是否相等，再分别取最小值
                if(str1[i-1] == str2[j-1])
                    res = dp[i-1][j-1];
                else
                    res = dp[i-1][j-1]+rc;

                dp[i][j] = min(res,temp1);
            }

        }
        return dp[n][m];
    }
};


int main()
{
    string str1 = "abc";
    string str2 = "adc";
    int ic = 5;
    int dc = 3;
    int rc = 100;
    Solution s;
    int res = s.solution(str1,str2,3,3,ic,rc,dc);
    cout<<res<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>