**暨南大学本科实验报告专用纸**

课程名称 算法分析与设计实验 成绩评定

实验项目名称 硬币问题 指导教师 李展 实验地点 N116

实验项目编号 3-0 实验项目类型 综合型

学生姓名 张瑞鹏 学号 2020101124

学院 信息科学技术学院 系 计算机 专业 计算机科学与技术

实验时间 2022 年 月 日 午～ 月 日 下 午

1. **问题描述**

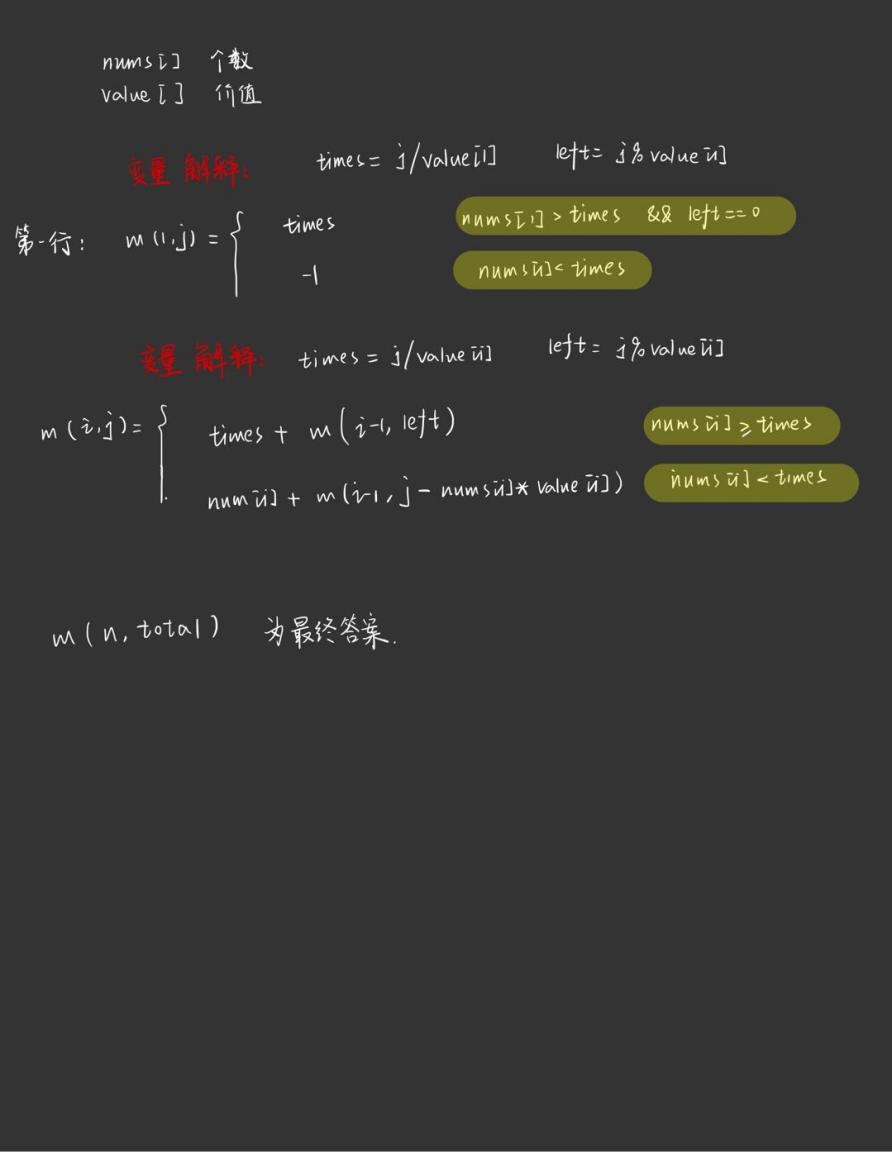
**设有n种不同面值的硬币，各硬币的面值存在于数组T[1:n]中。现要用这些面值的硬币来找钱。可以使用的各种面值的硬币个数存于数组Coins[1:n]中。对任意钱数0m20001，设计一个最少硬币找钱m的方法。**

**（二）算法思路**（用文字简单说明）

有两个数组分别记录了硬币的价值还有数量，我在代码中就直接定义为value,和nums分别代表。

假设面值数组是有升序的，如果没有将其排序至升序。

设置m（i,j）表示能用序号为1~i的硬币种类，需要组合成金钱的总数是j。因为我觉得找钱的话不能多也不能少，比如13块，就不能用10块和5块来凑成，需要用到面值比较小的金额来完成，所以用到类似背包问题逆过来的方式，从面值比较小的钱开始。



**这是我们列出的动规方程，找到了最优子结构。因为当前金额是当前最大的金额，所以尽可能的多用，用times = j/value[i],表示如果使用当前面值的货币，最多能使用多少张，用times = j % value[i]表示如果使用当前面值的货币，会余下多少金额无法除尽。此时就要分两种情况讨论1.假如当前的金额货币的数量value能够满足需求times，该货币就使用times张，余数怎么处理呢？调用m（i-1，left），也就是调用比当前金额小的货币来满足。 2. 假如当前的金额货币的数量value不能够满足需求times，该货币就全部用完，余数怎么处理呢？首先余数需要重新确定，因为之前算的余数是假如够用的前提之下的。余数 = j-value[i]\*nums[i]，调用m（i-1，j-value[i]\*nums[i]），也就是调用比当前金额小的货币来满足。**

**那么出口自然就是面值最小的金额了，面值最小的金额就必须满足既能够保证拥有的张数必须大于需要的张数，其次还不能有余数，才能给m（1，j） = times直接赋值，否则为-1。**

**另外还需要另外设定一个二维数组remember，一一对应m数组位置，remember[i][j],表示在m（i，j）这种情况之下，i号货币最适合放入多少个，之后输出结果。**

**输出结果：**

**先检查最后一个位置m（n，total），如果为-1代表无解，自然也是没有结果可以输出了。**

**否则的话建立一个辅助数组大小为货币种类的个数n，倒序完成填写，然后正序输出。**

**i = n ,j = total;**

**i进入循环后每次减1，j每次减去上一行货币的金额\*上一行的个数**

**直到遍历到最后一行**

**（三）算法实施步骤和流程**（伪代码/流程图等方式描述）

int findlesscoins(int\* value, int\* nums, int total,int n) {

int\*\* dp = new int\* [11];

int\*\* remember = new int\* [11];

int\* traceback = new int[n+1];

for (int i = 0; i <= 10; i++) {

dp[i] = new int[total+1];

remember[i] = new int[total + 1];

}//定义二维矩阵

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

for (int j = 0; j <= total; j++) {

remember[i][j] = 0;

}

}

for (int i = 0; i <= 10; i++) {

dp[i][0] = 0;

}//初始化首列元素

for (int j = 1; j <= total;j++) {//初始化首行元素

int times = j / value[1];

int left = j % value[1];

if (times <= nums[1] && left == 0) { dp[1][j] = times; remember[1][j] = times; }

else { dp[1][j] = -1; remember[1][j] = 0; }

}

for (int i = 2; i <= n; i++) {

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

int times = j / value[i];

int left = j % value[i];

if (times <= nums[i]) {//够的，按需求拿

if (dp[i - 1][left] == -1) dp[i][j] = -1;

else {

dp[i][j] = times + dp[i - 1][left];

remember[i][j] = times;

}

}

else {//不够全部拿完

if(dp[i - 1][j-nums[i]\*value[i]] == -1) dp[i][j] = -1;

else {

dp[i][j] = nums[i] + dp[i - 1][j - nums[i] \* value[i]];

remember[i][j] = nums[i];

}

}

}

}

if (dp[n][total] != -1) {

int j = total;

for (int i = n; i > 0; i--) {

traceback[i] = remember[i][j];

j -= value[i] \* remember[i][j];

}

cout << "分别的选取结果" << endl;

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

cout << traceback[i] << " ";

}

}

return dp[n][total];

}

**（四）源代码**（通过了编译运行的正确程序）

#include<iostream>

using namespace std;

int findlesscoins(int\* value, int\* nums, int total,int n) {

int\*\* dp = new int\* [11];

int\*\* remember = new int\* [11];

int\* traceback = new int[n+1];

for (int i = 0; i <= 10; i++) {

dp[i] = new int[total+1];

remember[i] = new int[total + 1];

}//定义二维矩阵

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

for (int j = 0; j <= total; j++) {

remember[i][j] = 0;

}

}

for (int i = 0; i <= 10; i++) {

dp[i][0] = 0;

}//初始化首列元素

for (int j = 1; j <= total;j++) {//初始化首行元素

int times = j / value[1];

int left = j % value[1];

if (times <= nums[1] && left == 0) { dp[1][j] = times; remember[1][j] = times; }

else { dp[1][j] = -1; remember[1][j] = 0; }

}

for (int i = 2; i <= n; i++) {

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

int times = j / value[i];

int left = j % value[i];

if (times <= nums[i]) {//够的，按需求拿

if (dp[i - 1][left] == -1) dp[i][j] = -1;

else {

dp[i][j] = times + dp[i - 1][left];

remember[i][j] = times;

}

}

else {//不够全部拿完

if(dp[i - 1][j-nums[i]\*value[i]] == -1) dp[i][j] = -1;

else {

dp[i][j] = nums[i] + dp[i - 1][j - nums[i] \* value[i]];

remember[i][j] = nums[i];

}

}

}

}

if (dp[n][total] != -1) {

int j = total;

for (int i = n; i > 0; i--) {

traceback[i] = remember[i][j];

j -= value[i] \* remember[i][j];

}

cout << "分别的选取结果" << endl;

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

cout << traceback[i] << " ";

}

}

return dp[n][total];

}

int main(void)

{

int n = 6;

int value[7] = {0,1,2,3,4,5,6};

int nums[7] = {0,10,7,4,8,7,4};

int total = 47;

cout << "test example:" << endl;

cout << n << endl;

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

cout << value[i] << " " << nums[i] << endl;

}

cout << total << endl;

int result = findlesscoins(value, nums, total, n);

if (result == -1) cout << "no result" << endl;

else {

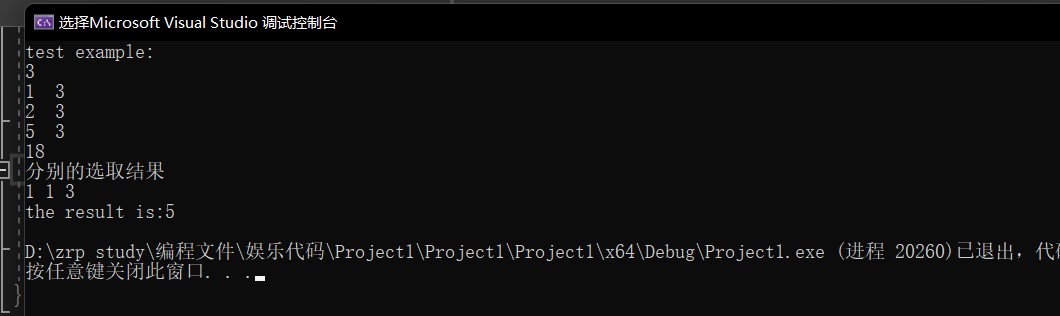
cout <<endl<< "the result is:" << result<<endl;

}

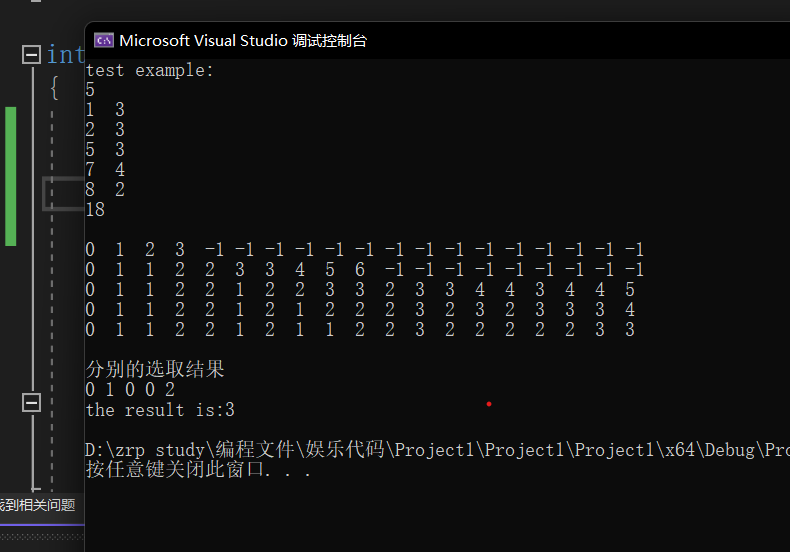
return 0;

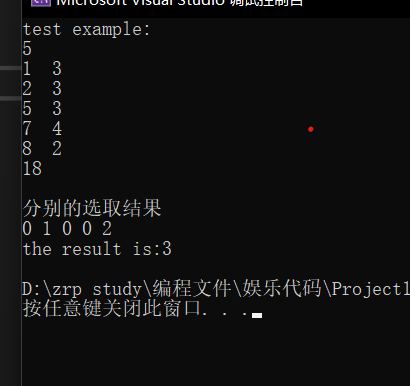
}

1. **测试结果**（至少有两个以上算例及程序运行结果，截图贴进实验报告）









**（六）实验总结**（至少三句话，可以写复杂度分析、遇到问题、可能的改进措施、心得体会等）

1．时间复杂度为主要来源于填写二维矩阵，时间复杂度为O(n\*total)

2.要先实现动规方程的分析，才能更容易的完成代码

3.要学会举一反三，即使问题做出改变，也不难看出这也是背包问题，应该能够顺利解答。

**暨南大学本科实验报告专用纸(附页)**

对于确定的d,j，k1是常数，故提取出外面

为了更直观看出这样转化的原理，看下面例子

f[i][ 6v+d ] = max( f[i-1][6v+d] ,f[i-1][5v+d] + w ,f[i-1][4v+d] +2w , f[i-1][3v+d] +3w.....)

应该没什么毛病，对于确定的j=6v+d,往里面塞0个，1个，2个..的价值

然后，我们对max括号里每一项，全部减去 6w ,6指的是当前j能装下的最多数目

f[i][ 6v+d ] = max( f[i-1][6v+d]-6w ,f[i-1][5v+d] -5w ,f[i-1][4v+d] -4w , f[i-1][3v+d] -3w....) +6\*w

同时减去相同数字，大小关系不变。

我们惊奇发现，这不就是上面的式子吗？k1=6 ,(k1-k)分别是那些5,4,3,那些系数

f[i][j]= max( f[i-1][ (k1-k)\* v[i]+d] -(k1 -k) \*w[i] ) + k1\*w[i] 0<=k<=cnt[i]

我们继续来看下面这个式子

f[i][ 6v+d ] = max( f[i-1][6v+d]-6w ,f[i-1][5v+d] -5w ,f[i-1][4v+d] -4w , f[i-1][3v+d] -3w....) +6\*w

它就告诉了我们，想知道左边的，必须求出右边括号里的，然后再加上6\*w，你可能会说，这不显然吗？

哎别，如果我们一个一个·枚举，效率不知道会低多少，这就引出了我们的杀手锏--单调队列

它的精华在于维护一个区间（窗口）的一个最值。

那么这个公式和窗口有什么关系呢？

嘿嘿，看清楚了，别眨眼，我们把(k1-k)用k,代换 ，修改k的定义域

原来是0<=k<=cnt[i] ，现在 k1-cnt[i]<= k <= k1 。这时候我们惊奇的发现，当j变化时，k1随之变化，k值也随之变化，但k值始终在一个窗口里！！

所以，我们对一个j，用一个单调队列维护窗口内的一些值，那些值？

接着看那个式子

f[i][j]= max( f[i-1][ (k1-k)\* v[i]+d] -(k1 -k) \*w[i] ) + k1\*w[i] 0<=k<=cnt[i]

(k1-k)-->k

f[i][j]= max( f[i-1][ k\* v[i]+d] -k \*w[i] ) + k1\*w[i] k1-cnt[i]<= k <=k1

我们维护窗口内, f[i-1][k\*v[i]+d] -k\*w[i]

到达我们滑动到目标j,的时候，区间最值已经出来了，f[i][j]就等于那个最值 + k1\*w[i]

大功告成！！

但我们还不满足，我们不满足于二维，我们把它滚成一维，如果对这方面还不熟悉，可以参考背包九讲。

首先，对于新输入的一个物品，w,v,c代表价值，体积，数目，已知最大体积V，那么它最多能容纳V/v 个，

int v,w,c;

cin>>v>>w>>c;

int k=V/v;

c=min(c,k);

c可能大于最大容量，所以取最小值.

我们枚举余数d, 对于每个体积,它模v,余数在0---v-1范围内

for(int d=0;d<v;d++)

然后对于一个确定的d,j最多能容纳

k=(V-d)/v;

枚举k,之前别忘了初始化单调队列

每次加入单调队列的，是dp[d+j\*v]-j\*w ， 这样，在计算每个目标点时，只需要提取出窗口最大值，再加上特定的k1\*w即可

for(int d=0;d<v;d++){// c表示最多能放多少个

head=tail=0;

k=(V-d)/v; 然后对于一个确定的d,j最多能容纳，放的个数，最多放多少个

for(int j=0;j<=k;j++){

while(head<tail&&dp[d+j\*v]-j\*w>=q2[tail-1])//把那些小的全部弹出

tail--;

q[tail]=j;//记录的是放多少个,记录是放了多少个的

q2[tail++]=dp[d+j\*v]-j\*w;

while(head<tail&&q[head]<j-c)//滑动窗口，前面有不满足的删掉

c = min{nums,times}

++head;//保证窗口最前面的既是最优值，而且还符合条件

dp[d+j\*v]=max(dp[d+j\*v],q2[head]+j\*w);//再这两个之间取一个最优值

}

}

}

最后输出dp[V]即可

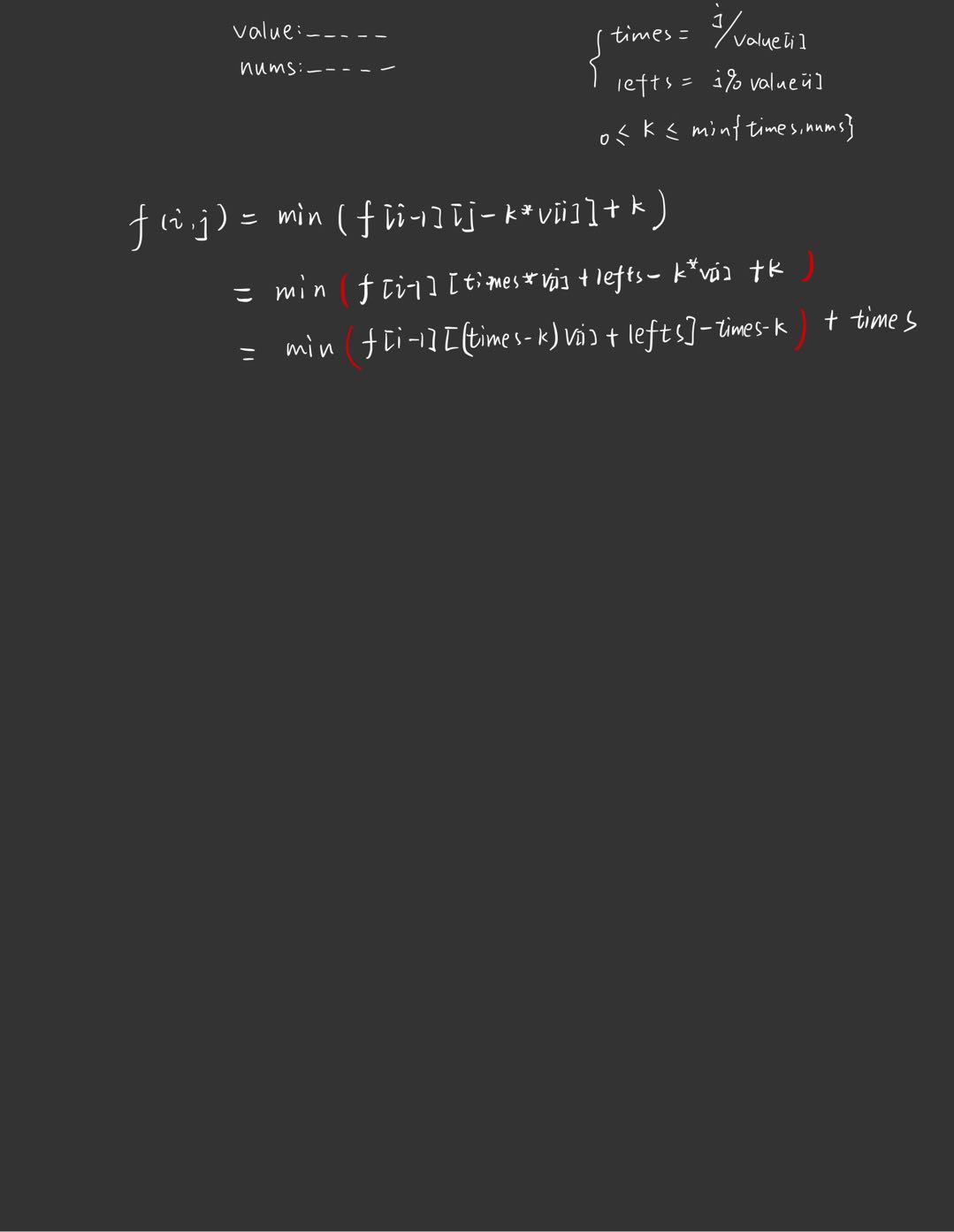
大功告成，如果对您有利，请点个赞，关注一下俺。

自我介绍一下，目前在读中国石油大学大一，平时多更新codeforces题解，热爱编程。

————————————————

版权声明：本文为CSDN博主「石油生产队里的秦三」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：<https://blog.csdn.net/jisuanji2606414/article/details/123384333>



我们现在想尝试一下对算法的时间复杂度进行优化：

首先说明一下变量：

F[i][j]表示的是可取前i种货币的最少个数

Times = j/values[i];这个表示的是当前金额和当前的容量，该种类型的货币最多能取多少

left =j%values[i];这是上一条的余数部分

这个就是我们的动态规划方程：

f[i][j]=min( f[i-1] [j-k\*v[i]] +k) 其中0<=k<=min{times，nums};

我们做一些公式变形：

J = left+times\*v[i]

于是就有了：

首先代替：

f[i][j]=min( f[i-1] [ left+times\*v[i] -k\*v[i]] +k) 其中0<=k<=min{times，nums};

合并同类项：

f[i][j]=min( f[i-1] [ (times-k)\*v[i]+ left] +k) 其中0<=k<=min{times，nums};

配凑目的是能把一些常量提到min外部

f[i][j]=min( f[i-1] [ (times-k)\*v[i]+ left] +times-k+times)

f[i][j]=min( f[i-1] [ (times-k)\*v[i]+ left] +times-k) +times

这就是变形之后的最终结果。

这样做的话可以看到是把余数相同的放在一类了，如果根据同余的j（横坐标）来遍历的话，前面遍历到的结果，后面是能用得上的。因此考虑用一个数组来存放这些内容。

但是考虑到是想要取最优的值，因此考虑到用单调队列存放结果，保护其中的最大值放在队头。

这样一来，就可以在相同元素的前提之下，依次后移，每次后到下一个同余的j，直到完成整张二维表的填写。当然这里也可以压缩成一维的形式，思路都是一样的，就不再补充。

时间复杂度：O（n\*m）我们可以直观的发现，尽管填表是跳着填的，但是二维数组每个位置都只遍历了一次，所以是O（n\*m）。n表示的是硬币种类的个数，m是硬币的总金额。

for(int d=0;d<value[i];d++){//遍历所有的余数，余数的范围当然是不能比value[i]大

head=tail=0;//空队列

k=(total-d)/v;//余数为d的有多少个

for(int j=0;j<=k;j++){

while(head<tail&&dp[i-1][d+j\*value[i]]-j<=q2[tail-1])

tail--;

q[tail]=j;

q2[tail]=dp[d+j\*value[i]]-j;//放进队尾

tail++;

while(head<tail&&q[head]<j-c)//

++head;

dp[i][d+j\*v]=min(dp[i-1][d+j\*v],q2[head]+j);

}

}

}

