-----Edit by ZhuSenlin HDU

本文说是《编程之美》2.18 新思路，其实也是 July 的《微软等公司面试 100 题》上的 32 题的解法。

两个序列大小均为 n，序列元素的值为任一整数，无序；
要求通过交换两个序列的元素，使序列 a 元素之和与序列 b 的元素之和的差最小（可能存在很多种组合，要求找出其中一种即可）。
如序列：1 5 7 8 9 和序列 6 3 11 20 17 我们可以通过交换得到新的序列 1 5 9 8 20 和序列 7 6 3 11 17，前者和伟 43，后者和为 44，两者之差为 1 最小。

题记：这道题和《编程之美》一书中 2.18 节的数组分割区别不大，但本人觉得《编程之美》这一节讲的不够透彻，不好理解（或许本人愚钝  **），故给出自己的思路，同时也给出打印其中一种方案的方法（这一点《编程之美》并没有提到）。**

首先对于两个素个数相等的序列 a、b 我们利用《编程之美》2.18 节的思想，将其合并为一个序列便于后续操作，序列中有负数的情况我们可以先预处理一下，让每个元素都加上一个初始值使得最后每个元素都为正。整个问题就转化为在一个元素个数为 2n 的正数数组中找出其中 n 个元素，使得这 n 个元素之和与剩下元素之和的差最小。

《编程之美》2.18 解法二中提到，从 2n 个数中找 n 个元素，有三种可能：大于 Sum/2，小于 Sum/2 以及等于 Sum/2。而大于 Sum/2 与小于等于 Sum/2 没区别，故可以只考虑小于等于 Sum/2 的情况，这一点我们仍然沿用这个思想。

下面谈谈变化的东西：

同样，利用动态规划的思想：

先给一个空间复杂度为 O(2N*N*Sum/2)即 O(N^2Sum)的方法，下面会对空间复杂度进行优化：

设 F[i][j][k]表示前 i 个元素中选取 j 个元素，使得其和不超过 k 且最接近 k。那么可以根据第 i 个元素是否选择来进行决策

状态方程如下：

$$F[i][j][k]=\max_{0<i\le 2n,\,0<j\le \min(i,n),\,0<k\le sum/2}\begin{cases}F[i-1][j][k], & \text{第i个元素不选}\\ F[i-1][j-1][k-A[i]]+A[i], & \text{第i个元素选（k}\ge \text{A[i])}\end{cases}$$

1-1

其中，F[i-1][j][k]表示前 i-1 个元素中选取 j 个使其和不超过但最逼近 k；

F[i-1][j-1][k-A[i]]在前 i-1 个元素中选取 j-1 个元素使其和不超过但最逼近 k-A[i]，这样再加上 A[i]即第 i 个元素就变成了选择上第 i 个元素的情况下最逼近 k 的和。而第一种情况与第二种情况是完备且互斥的，所以需要将两者最大的值作为 F[i][j][k]的值。

伪代码如下：

[cpp] view plaincopy

```
F[][][]← 0

for i ← 1 to 2*N

    nLimit ← min(i,N)

    do for j ← 1 to nLimit

        do for k ← 1 to Sum/2

            F[i][j][k] ← F[i-1][j][k]

            if (k >= A[i] && F[i][j][k] < F[i-1][j-1][k-A[i]]+A[i])

                then F[i][j][k] ← F[i-1][j-1][k-A[i]]+A[i]

return F[2N][N][Sum/2]
```

当然，前面已经提到，要给出一种方案的打印，下面我们谈谈怎么打印一种方案。

可以设置一个三维数组 Path[][][]来记录所选择元素的轨迹。含路径的伪代码如下，只是在上述伪代码中添加了一点代码而已。

[cpp] view plaincopy

```
F[][][]← 0

Path[][][]← 0

for i ← 1 to 2*N

    nLimit ← min(i,N)

    do for j ← 1 to nLimit

        do for k ← 1 to Sum/2

            F[i][j][k] ← F[i-1][j][k]

```

```
15.             if (k >= A[i] && F[i][j][k] < F[i-1][j-1][k-A[i]]+A[i])
16.
17.                 then F[i][j][k] ← F[i-1][j-1][k-A[i]]+A[i]
18.
19.                     Path[i][j][k] ← 1
20.
21. return F[2N][N][Sum/2] and Path[][][]
```

根据求得的 Path[][][]我们可以从 F[2N][N][Sum/2]往 F[0][0][0]逆着推导来打印轨迹对应的元素。伪代码如下：

[cpp] view plaincopy

```
1.  i ← 2N
2.
3.  j ← N
4.
5.  k ← Sum/2
6.
7.  while (i > 0 && j > 0 && k > 0)
8.
9.      do if(Path[i][j][k] = 1)
10.
11.         then Print A[i]
12.
13.             j ← j-1
14.
15.             k ← k-A[i]
16.
17.     i ← i-1
```

上面的伪代码的意思是，每当找到一个 Path[][][]=1，就将其对应的 A[i]输出，因为已经确定一个所以 j 应该自减 1，而 k 代表总和，所以也应该减去 A[i]。至于为什么不管 Path[][][]是否为 1 都需要 i 自减 1，这一点可以参照本人博文《背包问题——“01 背包”详解及实现（包含背包中具体物品的求解）》中的路径求法相关内容。

下面开始优化空间复制度为 O(N*Sum/2)

我们观察前面不含路径的伪代码可以看出，F[i][j][k]只与 F[i-1][][]有关，这一点状态方程上也能反映出来。所以我们可以用二维数组来代替三维数组来达到降低空间复杂度的目的。但是怎么代替里面存有玄机，我们因为 F[i][j][k]只与 F[i-1][][]有关，所以我们用二维数组来代替的时候应该对 F[i][j][k]的“j”维进行逆序遍历。为什么?因为只有这样才能保证计算 F[i][j][k]时利用的 F[i-1][j][]和 F[i-1][j-1][]是真正 i-1 这个状态的值，如果正序遍历，那么当计算 F[][j][]时，F[][j-1][]已经变化，那么计算的结果就是错误的。

伪代码如下

[cpp] view plaincopy

```
F[][]← 0

for i ← 1 to 2*N

    nLimit ← min(i,N)

    do for j ← nLimit to 1

        do for k ← A[i] to Sum/2

            if (F[j][k] < F[j-1][k-A[i]]+A[i])

                then F[j][k] ← F[j-1][k-A[i]]+A[i]


return F[N][Sum/2] and Path[][][]
```

上面的伪代码基本上和《编程之美》2.18 节最后所给的代码基本一致了，但是里面并不含 Path，如果要打印其中一种方案，那么仍需要 2N*N*Sum/2 的空间来存放轨迹。即

[cpp] view plaincopy

```
F[][]← 0

Path[][][]← 0

for i ← 1 to 2*N

    nLimit ← min(i,N)

    do for j ← nLimit to 1

        do for k ← A[i] to Sum/2

            if (F[j][k] < F[j-1][k-A[i]]+A[i])

                then F[j][k] ← F[j-1][k-A[i]]+A[i]

                Path[i][j][k] ← 1

return F[N][Sum/2] and Path[][][]
```

打印路径的伪代码与之前的一模一样，这里不再重写。

下面给出《编程之美》2.18 节所讲的“数组分割”中给出的数据进行**本文思想**的 C++代码实现数组 1　5　7　8　9　6　3　11　20　17 一共 10 个数，拆成两个数组，使得这两个数组和之差最小。

[cpp] view plaincopy

```cpp
#include <iostream>
#include <cstring>
#include "CreateArray.h"    //该头文件是动态开辟及销毁二维三维数组的，读者自己实现
using namespace std;
```

//这里参数 array 为整个合并后的数组序列，nLen 为合并后的数组长，nToBeClosed 是之前所提的 Sum/2

//算法时间复杂度为 $O(N^2Sum)$，空间复杂度为 $O(N^2Sum)$

[cpp] view plaincopy

```cpp
int AdjustArray(int array[], int nLen, int nToBeClosed)
{
    int*** F = NULL;
    int*** Path = NULL;
    CreateThreeDimArray(F,nLen+1,nLen/2+1,nToBeClosed+1);        //创建三维数组，存放每一个状态
    CreateThreeDimArray(Path,nLen+1,nLen/2+1,nToBeClosed+1);    //创建三维数组，存放轨迹
    for(int i = 1; i <= nLen; i++)
    {
        int nLimit = min(i,nLen/2);
        for(int j = 1; j <= nLimit; j++)
        {
            for(int k = 1; k <= nToBeClosed; k++)
            {
                F[i][j][k] = F[i-1][j][k];
                if(k >= array[i-1])
                {
                    if(F[i][j][k] < F[i-1][j-1][k-array[i-1]]+array[i-1])
                    {
                        F[i][j][k] = F[i-1][j-1][k-array[i-1]]+array[i-1];
                        Path[i][j][k] = 1;
                    }
```

```
                }
            }
        }
    }

        //打印调整后的其中一个数组
    int i = nLen, j = nLen/2, k = nToBeClosed;
    while(i > 0 && j > 0 && k > 0)
    {
        if(Path[i][j][k] == 1)
        {
            cout << array[i-1] << "\t";
            k -= array[i-1];
            j--;
        }
        i--;
    }
    cout << endl;

    int nRet = F[nLen][nLen/2][nToBeClosed];
    DestroyThreeDimArray(Path,nLen+1,nLen/2+1); //销毁轨迹表
    DestroyThreeDimArray(F,nLen,nLen/2+1);  //销毁状态表
    return nRet;
}
```

//这里参数 array 为整个合并后的数组序列，nLen 为合并后的数组长，nToBeClosed 是之前所提的 Sum/2

//算法时间复杂度为 O(N^2Sum)，空间复杂度不含 Path 为 O(NSum/2)，含 Path 为 O(N^2Sum)

[cpp] view plaincopy

```
int Fun2(int array[], int nLen, int nToBeClosed)
{
    int** F = NULL;
    int*** Path = NULL;
    CreateTwoDimArray(F,nLen/2+1,nToBeClosed+1);        //创建二维状态表
    CreateThreeDimArray(Path,nLen+1,nLen/2+1,nToBeClosed+1);//创建三维轨迹表

    for(int i = 1; i <= nLen; i++)
    {
        int nLimit = min(i,nLen/2);
        for(int j = nLimit; j >= 1; j--)
        {
            for(int k = array[i-1]; k <= nToBeClosed; k++)
```

```
            {
                if(F[j][k] < F[j-1][k-array[i-1]]+array[i-1])
                {
                    F[j][k] = F[j-1][k-array[i-1]]+array[i-1];
                    Path[i][j][k] = 1;
                }
            }
        }
    }

         //打印调整后的其中一个数组
    int i = nLen, j = nLen/2, k = nToBeClosed;
    while(i > 0 && j > 0 && k > 0)
    {
        if(Path[i][j][k] == 1)
        {
            cout << array[i-1] << "\t";
            k -= array[i-1];
            j--;
        }
        i--;
    }
    cout << endl;

    int nRet = F[nLen/2][nToBeClosed];
    DestroyTwoDimArray(F,nLen/2+1); //销毁二维状态表
    DestroyThreeDimArray(Path,nLen+1,nLen/2+1); //销毁三维轨迹表
    return nRet;
}
```

测试代码

[cpp] view plaincopy

```
int main()
{
    int array[] = {1,5,7,8,9,6,3,11,20,17};
    int nSum = 0;
    for(int i = 0; i < sizeof(array)/sizeof(int); i++)
        nSum += array[i];
    int nToBeClosed = nSum/2;

    cout << Fun(array,sizeof(array)/sizeof(int),nToBeClosed) << endl;
    cout << Fun2(array,sizeof(array)/sizeof(int),nToBeClosed) << endl;
    return 0;
```

```
12. }
```

本文的很多思路都和背包问题相同，详见本人博文中有关背包的文章