## 一、平分遗产

#### 1.理解问题

John不幸病逝，希望将自己的财产尽可能均分给他的两个孩子。他的财产是一张张价值不同的魔卡，请你设计策略，将其财产发给他的两个孩子并且两个孩子的财产总和插值最小，并进行分析。

输入：魔卡数量n，每张魔卡的价值

示例：5 1 7 5 3 4

输出：

第一位孩子得到价值为10的遗产：

价值为3的第4张魔卡

价值为7的第2张魔卡

第二位孩子得到价值为10的遗产

#### 算法设计

这个问题满足最优子结构，类似于01背包问题，我们只需要关注其中一位孩子分得的遗产，使它尽可能接近于总遗产的一半。

1. **状态和选择**

状态：可选择的魔卡还有哪些、分得的遗产是多少

选择：把这个魔卡分给他，不把这个魔卡分给他

1. **定义dp数组**

dp[i][v]=0/1的含义为：

对于前i个物品，能凑出价值为v的遗产--1

对于前i个物品，不能凑出价值为v的遗产--0

根据此定义，**base case为dp[0][]=0和dp[][0]=1**。（一开始搞错了，不管用几张卡都可以凑出价值=0的财产，也就是都不分给第一个孩子）

我们想计算的结果是从最右面第一个不全为0的列（对应的v表示第一位孩子分得的遗产总价值）开始，从上到下寻找第一个1处代表的魔卡分配方式。

1. **状态转换方程**

对于当前魔卡card[i-1]，

如果不选，则dp[i][j]=dp[i-1][j]；

如果选，则dp[i][j]=dp[i-1][j-card[i-1]]；

1. **回溯输出**

从最后一列开始向左寻找，直到找到第一个不全为0的列，对应的j就是该孩子分配的总魔卡价值；找到该列从上到下第一个1，对应的dp[i][j]=1，i就是他分配的魔卡，card[i]是这张魔卡的价值；继续查找dp[][j-card[i]]这列中第一个为1的 dp[i][j]，依次查找，直到查找到basecase即j=0。

#### 算法代码与分析

#include <stdio.h>

#include <math.h>

int dp[100][100]={0};

int n;

int check(int colomn)

{

    for (int i = 0; i <= n; i++)

    {

        if (dp[i][colomn] != 0)

        {

            return i;  //找到该列第一个不为0的row

        }

    }

    return -1; //没找到，该列全都是0

}

int main()

{

    printf("请输入魔卡总数量：");

    scanf("%d", &n);

    printf("请输入每个魔卡的价值：");

    int card[100];

    int total=0;

    for(int i = 1; i <= n; i++)

    {

        scanf("%d", &card[i]);

        total += card[i];

    }

    int want=total/2;

    //初始化base case

    for (int i = 0; i <= n; i++) {

        dp[i][0] = 1; // 和为0的情况，总是可以用空集合达成

    }

    for (int j = 1; j <= want; j++) {

        dp[0][j] = 0; // 没有任何卡片时无法组成正数

    }

    //遍历，状态转换

    for (int i = 1; i <= n; i++)

    {

        for (int j = 1; j <= want; j++)

        {

            if (j>=card[i])  //可以选择放

            {

                dp[i][j]= dp[i-1][j] || dp[i-1][j - card[i] ] ;

            }else{   //不能放

                dp[i][j]=dp[i-1][j];

            }

        }

    }

    for (int i = 0; i <= n ; i++)

    {

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

        {

            printf("%d ",dp[i][j]);

            if (j==want)

            {

                printf("\n");

            }

        }

    }

    int first = 0;//第一个孩子的遗产

    int colomn;

    //回溯输出

    for (colomn = want; colomn > 0 ; colomn--)

    {

        int i=check(colomn);

        if (i==-1)continue;

        else{

            first = colomn;

            printf("第一个孩子得到价值为%d的遗产:\n",colomn);

            printf("价值为%d的第%d张魔卡\n",card[i],i);

            colomn-=card[i];

            break;

        }

    }

    if (n==0)

    {

        printf("第一个孩子得到价值为0的遗产:\n");

    }

    while (colomn>0)

    {

        int i=check(colomn);

        printf("价值为%d的第%d张魔卡\n",card[i],i);

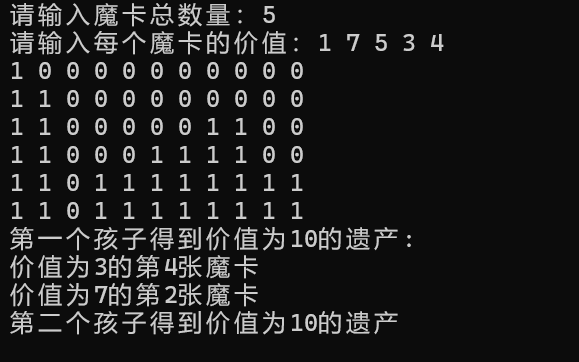
        colomn-=card[i];

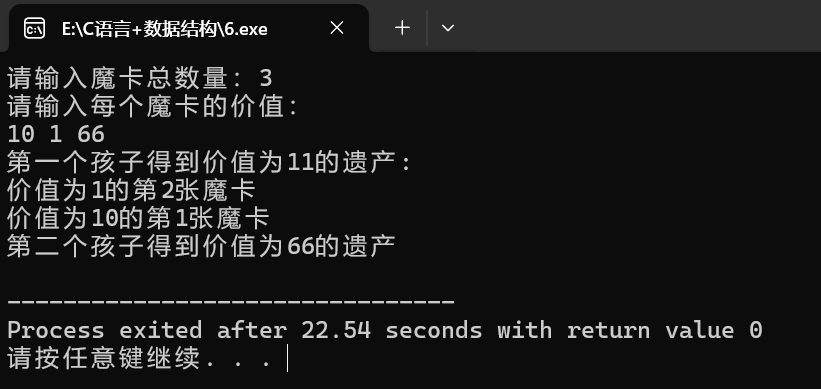
    }

    printf("第二个孩子得到价值为%d的遗产\n",total-first);

}

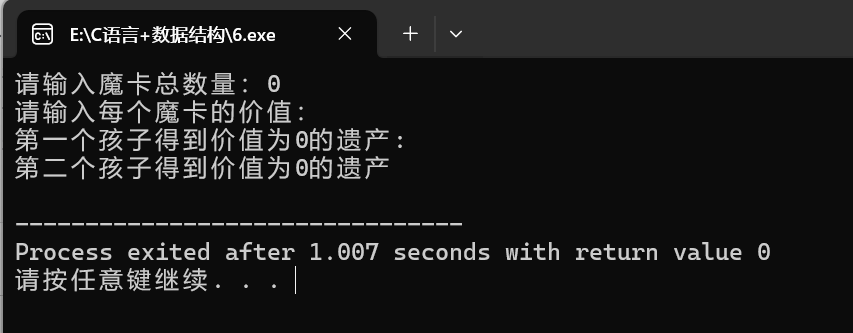
测试样例：







边界情况：



#### 性能分析

n为总的魔卡数量，total为总的魔卡价值

时间复杂度：主要开销在穷举计算dp数组上，为O（n\*total/2）；

空间复杂度：使用dp二维数组，为O（n\*total/2）；

#### 思考与改进

当魔卡的价值比较大、或魔卡的数量比较多的时候，时间复杂度和空间复杂度的开销都增加的很快，这时候可以考虑使用贪心算法，通过求局部的最优解，得到整体的近似最优解。

思路：将魔卡价值从大到小排序，每次将新的魔卡分给总价值最小的一组。

时间复杂度为：排序开销+遍历分财产开销 O(n\*logn+n)，

空间复杂度为：card数组，两个辅助数组表示两个孩子分到哪张魔卡，O(3\*n)

## 二、芯片连接

#### 1.理解问题

在二端口（in-out）芯片中，一组带编号in引脚与一组带编号out引脚分列两侧，不管是in引脚还是out引脚，编号都可能重复。假设仅允许具有相同编号的in引脚和out引脚相连（一次连接），请设计算法找到对于某芯片能够实施的最大连接数，连接的线不能交叉，并予以分析。

输入：分别输入in端引脚总数及编号序列，out端引脚总数与编号序列

示例：

8

1 2 1 3 5 4 6 9

8

2 3 5 1 5 4 6 1

输出：

芯片能实施的最大连接数为5

6-6 4-4 5-5 1-1 2-2

#### 2. 算法设计

由于连接的线不能交叉，这是一个最长公共子序列的问题LCS。

（1）Base case：lcs[0][]和lcs[][0]都为0

（2）dp数组定义：

Lcs[i][j]=m表示长度为i的子串s1和长度为j的子串s2的最长公共子序列的长度为m

1. 状态转移方程：  
   当s1[i]==s2[j]时，lcs[i][j]=lcs[i-1][j-1]+1 （左上角对角线+1）

当s1[i]!=s2[j]时，lcs[i][j]=max ( lcs[i-1][j] , lcs[i][j-1] ) （左边和上面取最大值）

1. 回溯

我们想求的是lcs[len1][len2]处的值，即为芯片能实施的最大连接数；

从此处开始回溯，

如果 s1[i]==s2[j]，说明从左上角对角线得来的，输出，i--,j--

Else s1[i] != s2[j]，

如果lcs[i][j]==lcs[i-1][j]，说明是从上面得来的，i--;

如果lcs[i][j]==lcs[i][j-1]，说明是从左面得来的，j--;

一直循环输出连接，直到i=0或者j=0。

#### 3.算法代码与分析

#include <stdio.h>

#include <math.h>

int main()

{

    int len1,len2;

    int s1[100]={0};

    int s2[100]={0};

    printf("请输入in端引脚总数:");

    scanf("%d",&len1);

    printf("请依次输入in端编号序列:");

    for(int i=1;i<=len1;i++)   //都是从下标开始存储

    {

        scanf("%d",&s1[i]);

    }

    printf("请输入out端引脚总数:");

    scanf("%d",&len2);

    printf("请依次输入out端编号序列:");

    for (int i = 1; i <= len2 ; i++)

    {

        scanf("%d",&s2[i]);

    }

    int lcs[100][100]={0};

    //初始化basecase

    for (int i = 0; i <= len1; i++)

    {

        lcs[i][0]=0;

    }

    for (int i = 0; i <= len2; i++)

    {

        lcs[0][i]=0;

    }

    //穷举 状态转移

    for (int i = 1; i <= len1; i++)

    {

        for (int j = 1; j <= len2; j++)

        {

            if(s1[i]==s2[j])

            {

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

            }

            else lcs[i][j]= lcs[i-1][j] > lcs[i][j-1] ? lcs[i-1][j] : lcs[i][j-1] ;

        }

    }

    //回溯

    printf("芯片能实施的最大连接数为%d\n",lcs[len1][len2]);

    int i=len1,j=len2;

    while (i>0&&j>0)

    {

        if(s1[i]==s2[j])

        {

            printf("%d-%d ",s1[i],s2[j]);

            i--;

            j--;

        }else {

            if (lcs[i][j]==lcs[i-1][j])i--;

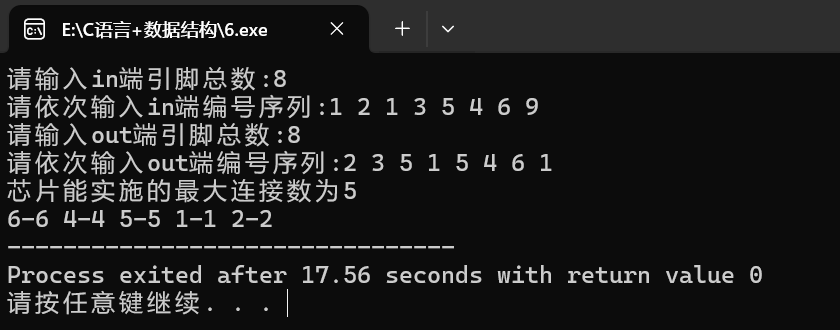
            if (lcs[i][j]==lcs[i][j-1])j--;

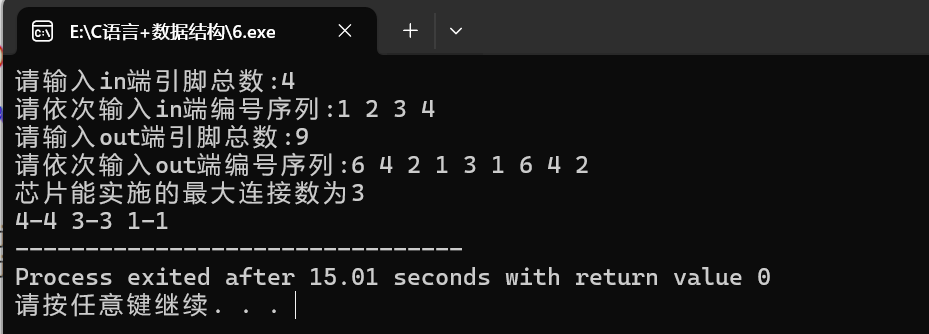
        }

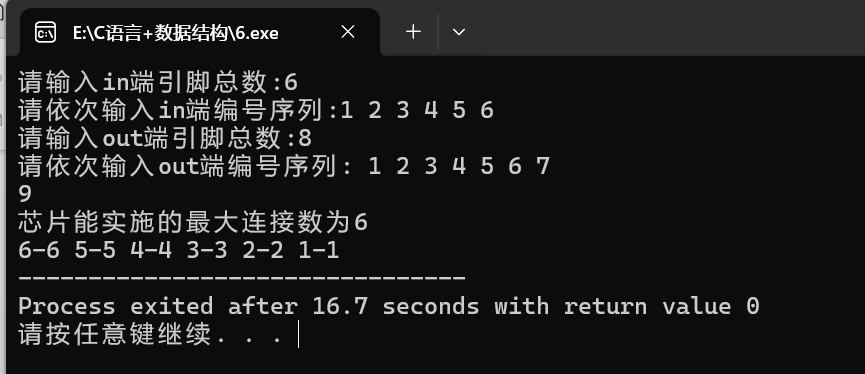
    }

}

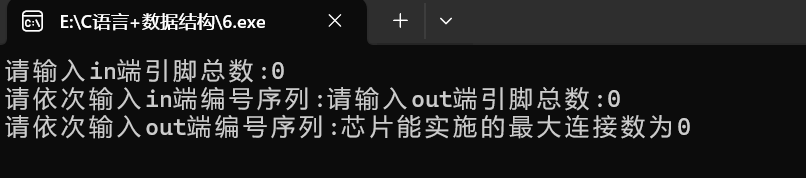
测试数据：







边界情况：



#### 4.性能分析

设in端、out端的引脚数量分别为len1、len2，

时间复杂度：读入两端编号O(len1+len2)+穷举状态O(len1\*len2)+回溯O(len1+len2)，为O(len1+len2+len1\*len2+len1+len2) = O(2\*(len1+len2)+len1\*len2)。

空间复杂度：存放端口编号的两个数组，lcs二维数组，为O(len1+len2+len1\*len2)。