程设II第四次OJ习题实验报告

姓名：林宸昊

学号：PB20000034

通过OJ题目的最大概率

题目描述

> 小明尝试了很多办法还是没法通过本周的上机题，他决定铤而走险。 他的室友是一名大佬，写出来的代码优雅且整洁，小明知道，如果他能抄到更多个字的室友的代码，就有更大的概率通过上机题。

> 由于室友的代码具有局部不可破坏性，不太聪明的小明只能挑取**一些行整行抄取**。但是提交的代码如果有**连着两行完全一样**，就会在代码查重环节被助教当场抓获并记零分。

> 请你帮帮小明：在不被当场抓获的前提下，挑选哪些行复制粘贴，才能抄到最多字呢?

> 番外:可惜的是小明最终仍然没有通过代码查重，同学们不要学习他。

题目要求

> 输入是以空格分割的一系列整数，每个数字代表该行代码有多少字 由于题目不难，代码不会超过100行，但也不会少于一行。另外，由于他的室友严格遵守代码规范要求，每行的字数n满足0<=n<=80。

> 输出为一个整数，代表在不会被助教当场抓获的情况下，小明最多能抄到多少字

如:

输入

```
10 24 13 5
```

输出

```
29
```

抄第二行和第四行最多 为29

具体思路

将输入的数据储存在两列中，如输入123456，储存情况为:

```
1 2
3 4
```

```
5 6
```

由于相邻数不能相加，则

> 第一列的数仅有与其下一排两列中的较大数相加的方式，

> 第二列的数仅有与其下一排第二列以及其下两排第一列中较大数相加的方式。

> 如2只需选择3,4之间较大数，选择5,6必可以额外选择3,4。

采用动态规划从下往上累加。结果不断累加。

具体设计

```
for (i = row - 1; i >= 1; i--)
{   //从最低层开始向上累加
    num[i][1] += max(num[i + 1][1], num[i + 1][2]);
    num[i][2] += max(num[i + 2][1], num[i + 1][2]);
}
```

由于从一开始有以第一行为起点与第二行为起点两种抄法，故最后输出两个起点中的较大值。

```
printf("%d", max(num[1][1], num[1][2])); //输出最终两种结果中较大者
```

实验过程

主要思路较为清晰且易于得出，主要是处理输入并转化为需要的格式，同时得到行数row。

```
while (j <= 2)
    {   //输入数据
        scanf("%d", &num[i][j]);
        if (j == 2)
        {
            j = 1;
            i++;
        }
        else
            j = 2;
        if (getchar() == EOF) //控制结束
            break;
    }
    row = i;
```

这样我们就解决了这个问题。

全排列输出

题目描述

> 给定一没有重复数字的序列，返回其所有可能的全排列。 输出需要按照输入的从前到后的顺序排列。具体参考样例数据。

题目要求

> 输入是用空格分割的几个数字，数字个数n满足1 <= n <= 9.

> 输出你的排列，每行一个，按照从小到大顺序。

如：

输入

```
2 1 3
```

输出

```
213
231
123
132
321
312
```

最后输出顺序取决于输入顺序而非数字本身大小。

具体思路

用一个数组`target[11]`储存得到的合法排列。

由于输出顺序与元素大小无关，则可将输入全部看做字符，按输入顺序储存在一个字符串中。本题采用指针`char* num`储存以适应不同大下的输入。

本题采用递归方法。如输入1 2 3 4，对1放在首位置的情况进行排列。而1开头的排列数等于12,13,14开头的排列数，即

```
w(1) = w(12) + w(13) + w(14)
     = w(123) + w(124) + w(132) + w(134) + w(142) + w(143)
     = ......
```

终止条件为候选数为0，即不存在剩余的可加入排列的数，则视为排列完成：

```
if(!rest) print(target), return;
```

对于所有的数，用循环按顺序遍历即可。

```
void output(char *arr, int pos)
{   /*
      arr储存当前正在进行排列的数据
      pos储存当前首先排列的数据所处的位置
    */
    int len = strlen(arr);
    //len储存当前正在排列的数据个数
    target[max_len - len] = *(arr + pos);
    //将pos位置的首先排列的数据放进目标字符串
    if (1 == len)
    {   //如果这是最后一个数据，输出排列，结束循环
        puts(target);
        return;
    }
    char *temp;
    //临时存放排除掉已排列数据的剩余数据
    temp = (char *)malloc(10 * sizeof(char));
    memcpy(temp, arr, 10);
    memcpy(temp + pos, temp + pos + 1, len - pos);
    //除掉目标数据
    temp[len] = 0;
    //多余位初始化
    for (int i = 0; i < len - 1; i++)
    {   //按从小到大顺序继续处理剩余数据
        output(temp, i);
    }
    free(temp);
}
```

实验过程

本题的难点在于如何传递参数使函数所接收到的候选数逐步减少。可以通过临时构建数组存放排除已存放数据的更新后数据，然后据此进行下一层递归。

```
memcpy(temp + pos, temp + pos + 1, len - pos);
```

实验总结

两道题较为基础。

前者主要考察动态规划，有些类似杨辉三角，难点在于如何构建模型；

后者主要考察递归，难点在于递归过程中的剪枝，即参数的变化。

对于全排列的选做部分，由于上述程序实质就是对输入数字的有序全排列，故只需按照输入的目标序号直接输出即可，思路大体一致，故不多赘述。