微信红包程序实验报告

孙嘉玺

22920162204038

2018 年 3 月 4 日

目录

1 题目描述

微信红包程序：给定一个钱数 m（整数），发红包人数 n，将钱数拆成 k 个指定的吉利数，并发出。

2 算法设计

这个题目有两种算法实现，第一种是搜索算法，即穷举出每种可能性，需要用递归实现算法复杂的为

$$O_{(k^m)} or O_{(m^k)}$$

因此这种算法的复杂太高，当我们有 100 个红包，20 个吉利数的时候，计算法显然无法在可以容忍的时间内完成，因此需要一种复杂度较低，比较高效的算法。经过一番思考我设计的算法如下定义一个布尔函数

$V(i,j)$,i 表示当前分配好的红包数目，j 表示当前这些分配好的红包数目的总钱数，$V(i,j)$ 的值就表示这种情况能不能实现，如果可以就为 $True$，否则就为 $False$. 每个吉利数的数值为 lv_i,$i = 1, 2, \ldots, k$. 而

$$V(i,j) = \sum_{i=0}^{k} V(i-1, j-lv_i)$$

初始化 $V(0,0) = True, V(0,j) = False$,$j$ 从 0 取到 m. 这样这个问题就能在三个循环下结局，所以复杂的为 $O_{(n*m*k)}$ 从而可以用计算机很快解决

3 代码实现

3.1 实验环境

- C 语言
- gcc 编译器
- terminal 中运行程序

3.2 实验代码

```
#include <stdio.h>
#include <stdlib.h>

# define True  1
# define False 0
# define DIVISION_VALUE 100
# define LN 16
// 100 represent the DIVISION_VALUE is 0.01


int main() {
    int money_sum;
    int i, j, k;
    int luck_nums[LN] = {   60  , 66    , 600   , 660   ,
                            800 , 880   , 888   , 80    ,
                            360 , 6600  , 1600  , 1800  ,
                            500 , 2000  , 100   , 200
                        };

    int num_redp;
    scanf("%d %d", &money_sum, &num_redp); // money first
```

```
    money_sum = money_sum * DIVISION_VALUE;


    // init the search state and allocate the space
    int** dp;
    dp = (int **)malloc(sizeof(int *) * (num_redp + 1));
    for (i = 0; i <= money_sum; ++i) {
        dp[i] = (int *)malloc(sizeof(int) * (money_sum + 1));
    }

    printf("sucess allocate\n");
    // int dp[1000][1000] = {};
    // init
    for (i = 0; i <= num_redp; ++i) {
        for (j = 0; j <= money_sum; ++j) {
            dp[i][j] = False;
        }
    }
    dp[0][0] = True;


    // should keep the s-lucky_num > 0 if < 0 should allocate the False
    // start the search
    for (i = 1; i <= num_redp; ++i) {
        for (j = 1; j <= money_sum; ++j) {
            for (k = 0; k < LN; ++k) {
                // get the True or Fasle;
                if (j - luck_nums[k]>= 0 && dp[i - 1][j - luck_nums[k]]) {
                    dp[i][j] = True;
                    break;
                }
            }
        }
    }


    // collect the red bag
    int init_sum = money_sum;
    for(i=num_redp;i>0;--i){
```

```
        for(k=0;k<LN;++k){
            if(init_sum - luck_nums[k] >= 0 && dp[i-1][init_sum - luck_nums[k]]){
                init_sum = init_sum - luck_nums[k];
                printf("%d:%.2f ",i, luck_nums[k]/100.0);
                break;
            }
        }
    }
    printf("\n%d\n", dp[num_redp][money_sum]);
    return 0;
}
```

4 实验数据 & 结果

4.1 幸运数：

```
0.60 , 0.66  , 6.00  , 6.60  ,
8.00 , 8.80  , 8.88  , 0.80  ,
3.60 , 66.00 , 16.00 , 18.00 ,
5.00 , 20.00 , 1.00  , 2.00
```

4.2 钱数 & 红包数 & 分得红包

钱数	红包数	分得红包			
90	3	6	66	18	
100	15	12 * 0.6	8.8	66	18
20	9	6 * 0.6	6.6	8.8	1
17	21	20 * 0.6	1		
13	9	7 * 0.6	8	0.8	
9	4	0.6	6.6	8.0	1
500	40	内存错误			
7	100	failed!			

5 实验分析

5.1 优点

在较复杂的情况下如钱数为 100，红包数为 15，幸运数字有 16 个的情况下，暴力搜索的算法，是不能在客观的时间内完成的，而这种算法却可以在毫秒级别完成。

5.2 缺陷

这种算法也有弊端就是要消耗 $n \times m$ 的空间，比较占用内存。另一个缺陷在代码上，因为 c 语言掌握的不是很牢固，所以程序分配较大内存时不能再用 malloc 来实现，但是如果代码迁移到更高级的语言如 c++，Python 应该不会出现此问题

6 实验总结

当我们考虑用算法解决一个问题的时候，往往第一次就能想到的算法不会是一种很好的算法，需要我们更加深入的理解问题本身，并且查阅资料，研究别人的算法，别人的思路。这样，我们才能在思考中不仅解决了问题，更丰富了我们自己的知识。

7 参考文献

- 动态规划
- 子集和问题