算法设计与分析实验报告

微信红包程序

张富威 22920162204078

问题描述：

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

算法思路：

> 使用**递归穷举**的方法，与**八皇后**问题有点相似，找到结果就输出，否则回溯继续寻找，实现任意多种类（红包数额）的个数分配，用户可以选择找出多种分配方法或者一种分配方法，若没有结果，则输出“找不到”。算法复杂度与幸运红包种类数有关系，随着种类增加，成指数上升。

实现代码：

```
#include <stdio.h>
#include <stdlib.h>
#define MONEY 0 //MONEY表示对应红包类型的金额
#define SUM 1  //SUM表示对应数额红包的个数
#define MAXMONEYTYPE 10 //最大幸运红包的种类数

/*
 *输入幸运红包的种类数以及金额数
 */

int inputLuckyPocket(float pocket[][MAXMONEYTYPE]) {
    int amount_pocket; //红包种类数
    printf("请输入幸运红包的种类总数:");
    scanf("%d", &amount_pocket);
    printf("请输入幸运红包的具体面额，以空格隔开:");
    for (int i = 0; i < amount_pocket; ++i) {
        scanf("%f", &pocket[MONEY][i]);
    }
    return amount_pocket;
}


void PrintPocket(int amount_pocket, float pocket[][MAXMONEYTYPE]) {
    int i = 0;
    while(i < amount_pocket) {
        printf("%f \n", pocket[MONEY][i]);
        ++i;
```

```
    }
}

/*判断是否满足条件*/
int IsContent(int amount_people, float amount_money, int amount_pocket, float
pocket[][MAXMONEYTYPE]) {
    float sum_money = 0; //当前红包的总额
    int pocket_num = 0; //当前红包个数
    for (int i = 0; i < amount_pocket; ++i) {
        sum_money += pocket[MONEY][i] * pocket[SUM][i];
        pocket_num += pocket[SUM][i];
    }
    if(sum_money == amount_money && pocket_num == amount_people) { //amount_money
实际所需的红包总额  amount_people 实际所需的红包个数
        return 1;
    } else {
        return 0;
    }
}
void PrintResult(int amount_pocket, float pocket[][MAXMONEYTYPE]) {
    printf("----------------------------\n");
    for(int i = 0; i < amount_pocket; ++i) {
        printf("红包金额:%.2f,  个数: %1.0f\n", pocket[MONEY][i], pocket[SUM][i]);
    }
    printf("----------------------------\n\n");
}

/*
 *递归穷举寻找合适的红包配额
 *amount_people 总人数
 *amount_money 总钱数
 *amount_pocket 红包类型总数
 *type_pocket 当前递归层所处的红包类型序号
 */
int existRes = 0; //是否有结果的标志

int GetOneResOfNumber(int amount_people, float amount_money, int amount_pocket,
float pocket[][MAXMONEYTYPE], int type_pocket) {
    if(type_pocket > amount_pocket) {
        return 0;
    }
    int flag = 0;
    for(int i = 0; i <= amount_people; ++i) {
        pocket[SUM][type_pocket - 1] = i;
        type_pocket += 1;
        if(IsContent(amount_people, amount_money, amount_pocket, pocket)) {
            existRes = 1;
            PrintResult(amount_pocket, pocket);
            return 1;
```

```
        }
        flag = GetOneResOfNumber(amount_people, amount_money, amount_pocket,
pocket, type_pocket);
        if(flag) {
            return 1;
        }
        type_pocket -= 1;
    }
    pocket[SUM][type_pocket - 1] = 0;
    return 0;
}

/*寻找所有可能的结果*/
int GetAllResOfNumber(int amount_people, float amount_money, int amount_pocket,
float pocket[][MAXMONEYTYPE], int type_pocket) {
    static int count = 0;
    if(type_pocket > amount_pocket) {
        return 0;
    }
    int flag = 0;
    for(int i = 0; i <= amount_people; ++i) {
        pocket[SUM][type_pocket - 1] = i;
        type_pocket += 1;
        if(IsContent(amount_people, amount_money, amount_pocket, pocket)) {
            count++;
            existRes = 1;
            PrintResult(amount_pocket, pocket);
        }
        GetAllResOfNumber(amount_people, amount_money, amount_pocket, pocket,
type_pocket); //递归穷举
        type_pocket -= 1;
    }
    pocket[SUM][type_pocket - 1] = 0;
    return 0;
}


int main() {
    float amount_money; //红包总金额
    int amount_pocket; //红包金额种类数
    float pocket[2][MAXMONEYTYPE] = {}; //pocket[1]存放红包金额,pocket[2]存放对应金
额红包的个数
    int amount_people; //总人数(总红包数)
    int choose = 0;
    printf("--------------------------\n请输入总人数:");
    scanf("%d", &amount_people);
    printf("请输入红包的总额:");
    scanf("%f", &amount_money);
```

```
    amount_pocket = inputLuckyPocket(pocket);

    printf("请选择[找到所有结果:输入1,  找到一个结果:输入0]\n");
    scanf("%d", &choose);
    if(choose == 0) {
        GetOneResOfNumber(amount_people, amount_money, amount_pocket, pocket, 1);
    } else {
        GetAllResOfNumber(amount_people, amount_money, amount_pocket, pocket, 1);
    }
    if(!existRes) {
        printf("找不到!\n");
    }
    return 0;
}
```

优化与不足：

> 由于算法是从所有红包都为0个开始穷举，只是对**上限**进行了限制，要优化的话可以从n个红包（人数）开始穷举（**省去一开始从0穷举到n的时间**），从而实现时间上的优化，但是并没有改变时间复杂度。 还有**数据结构**存在缺点，没有定义一个像样的结构来存储相关信息，变量相对比较分散，若从数据结构上进行优化，可以增加代码的可读性以及效率。
>
> 在和同学的交流中，了解到可以使用动态规划来进行红包分配，具体的思路就是用递推的方法，从零开始，如果到最后一个红包的时候，在幸运红包里正好符合余下钱的种类时，说明有解，从而可以反向输出。这个算法把复杂度降到O(m* n *w)，缺点就是需要大量的辅助空间来做标记。

实验结果：

```
----------------------------
请输入总人数:200
请输入红包的总额:500
请输入幸运红包的种类总数:10
请输入幸运红包的具体面额,以空格隔开:1.66 2.33 3.33 5.55 6.66 1 2 3 4 5
请选择[找到所有结果:输入1,  找到一个结果:输入0]
0
------------------------------
红包金额:1.66,  个数: 0
红包金额:2.33,  个数: 0
红包金额:3.33,  个数: 0
红包金额:5.55,  个数: 0
红包金额:6.66,  个数: 0
红包金额:1.00,  个数: 0
红包金额:2.00,  个数: 100
红包金额:3.00,  个数: 100
红包金额:4.00,  个数: 0
红包金额:5.00,  个数: 0
------------------------------
```

结论：

> 在红包种类数为5个以内，没有明显的时间上的感觉，该算法与红包种类数和输入顺序都有关系，原因就是层层递增的算法总是要先跑到栈底在一层一层回溯，如果上述的输入顺序换成1 2 3 4 5在开头，有可能需要花费很长时间才能找到结果。
>
> 所以我的算法在幸运红包种类数较少的时候才能很快的给出结果。