组合数学项目文档

——售票处找零问题

同济大学 软件学院 15级2班 1552651 王依睿

- 使用说明
  - 操作手册
    - 开始
    - 输入购票人数
    - 求解
  - 注意事项
- 概述
  - 问题描述
  - 求解算法
    - 算法一 搜索策略
    - 算法二 栈模型
    - 算法三 递归算法
    - 算法四 递推算法
    - 算法五 组合算法
  - 文件目录
- 总结
  - 数据统计
  - 比较分析
- 实现
  - 算法函数

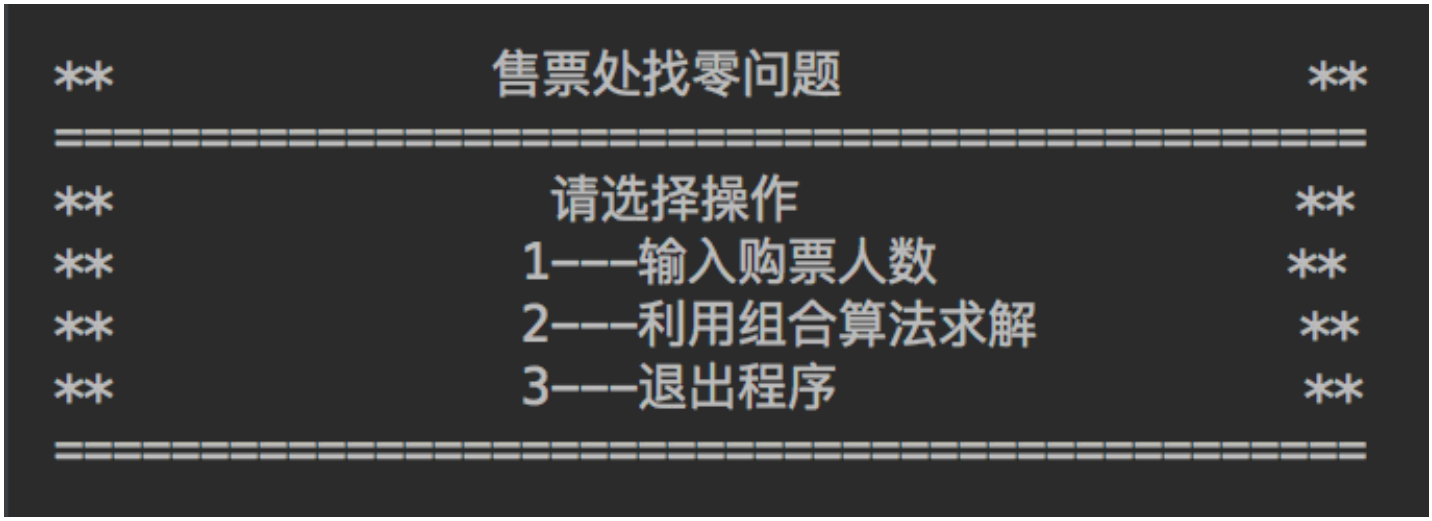
- 算法一 搜索策略
- 算法二 栈模型
- 算法三 递归算法
- 算法四 递推算法
- 算法五 组合算法

使用说明

操作手册

开始

运行程序后，输出的操作说明。。



输入购票人数

- 选择操作1。
- 请求用户输入购票人数。

求解

- 选择操作2。
- 输出当前算法当n为输入值时的方案数和所用时间。

```
请选择操作：2
n为15时，共有0种方案，所用时间为0.000003s
```

注意事项

- 所用时间与运行此程序的计算机性能相关，故在不同的计算机上运行从程序时，相同的购票人数和算法可能会出现时间不同的情况。
- 当随机数个数较大或计算机性能相对较弱时排序时间会较长，请耐心等待。
- 当购票人数较大时方案数会产生溢出。

概述

问题描述

农夫John和他朋友们去参加展览会。展览会的门票为$50。John发现一个奇怪的现象: 在排队购票的2n个人中，总有n个人拿的是面值$100的钞票，而另外的n个人拿的是面值为$50的钞票。农夫相知道的是在这种情况下这2n个人共有多少种排队方式，使得售票处不至于出现找不开钱的局面。假设售票处原本没有预备零钱的。

求解算法

算法一 搜索策略

用回溯法来直观的描述所有情况。算法制定 一个变量k用于记录售票处有$50钞票的张 数。初始时另k=0，收到一张$50钞票时k+1。若某人手持$100钞票且k=0时则回溯， 否则继续递归。若第2n个人购完票即递归 进行到第2n层时计数器累加1。递归结束后， 计数器中的值便为排队的方案数。

算法二 栈模型

分析:在任一时刻，若第n个人手持$100的钞票购票，则在此之前一定有不少于n个人手持$50的钞票购票。所以售票处收到的面 值为$50的钞票最终将全部找出，而所有收到的$100的钞票最终将全部留下。 用栈来表示这一过程:若某人手持$50的钞票购票，相当于一个元素进栈;若某人手持$100的钞票购票，相当于一个元素出栈; 问题转化成:若1~n个元素依次进栈，共有多少种出栈顺序。

算法三 递归算法

令f(m,n)表示m个人手持$50的钞票，n个人手持$100的钞票时共有的方案数。 1. n=0, 所有人都是拿的$50的钞票，所以f(m,0)=1 2. m<n, 把所有$50的钞票都找出去也会出现找不开钱的局面，所以f(m,n)=0 3. 其他情况

A. 第(m+n)个人手持$100的钞票，则在他之前的m+n-1个人中有m个人手持$50的钞票， n-1个人人手持$100的钞票，此种情况共有f(m,n-1) B. 第(m+n)个人手持$50的钞票，则在他之前 的m+n-1个人中有m-1个人手持$50的钞票，n个人手持$100的钞票，此种情况f(m,n)： 1. m<n时，f(m,n)=0 2. n=0时，f(m,n)=1 3. others，f(m,n)=f(m,n−1)+f(m−1,n)

算法四 递推算法

依然使用递归算法的状态转移方程，由初始条件向终止条件推导。

算法五 组合算法

从(0,0)点出发，对若遇到1就沿y轴正方向走一格，若遇到0就沿x轴正方向走一格， 每一种走法对应一条从(0,0)点到(n,n)点的 路径。所以所有满足条件的2n位二进制数 都是从(0,0)点对角线及对角线上方到(n,n) 点的路径。

文件目录

- _1552651_wangyirui_*n*.cpp（主文件）
- _1552651_wangyirui_*n*.exe（可执行文件）
- _1552651_wangyirui_Report.pdf（项目文档）

注：n为项目对应的算法号

总结

数据统计

	搜索策略	栈模型	递归算法	递推算法	组合算法	计算结果
n = 3	0.000003	0.000005	0.000007	0.000006	0.000006	5
n = 7	0.000026	0.000006	0.000015	0.000007	0.000006	429
n = 10	0.000461	0.000015	0.000338	0.000020	0.000004	16796
n = 13	0.019743	0.000016	0.015480	0.000020	0.000006	742900
n = 15	0.132752	0.000008	0.097399	0.000029	0.000004	9694845
n = 17	1.433787	0.000009	1.010881	0.000030	0.000006	129644790
n = 19	19.040404	0.000008	13.668176	0.000030	0.000006	6564120420
时间复杂度	$O(2^n)$	$O(n^2)$	$O(2^n)$	$O(n^2)$	$O(n)$	

比较分析

- 当n较小时，时间复杂度的常数系数c相对于n的阶数占主要地位，五种算法的所用时间相差较小，当n增大时五种算法的时间复杂度n阶数与其所用时间的正向关系逐渐体现明显。
- 搜索策略通过深度优先搜索和回溯的方法遍历所有的可行情况，优点是可以得到所有的可行情况并加以记录输出，缺点$O(2^n)$的时间复杂度使得运行的时间较长。
- 递归算法思路和代码实现较为简单，但由于递归函数在计算后项时调用前项的递归函数，存在2个至k个前项的单独调用，当n较大时，处于较前的前项k值很大。随着n的增大，冗余计算的增长时爆炸性的，时间复杂度达到$O(2^n)$。
- 递推算法改进了递归算法的不足，依然使用递归算法的状态转移方程，由初始条件向终止条件推导记录。从前面的项开始计算并记录当前项的数据以用于后面的项的相加计算，从而避免了冗余计算带来的巨大的时间开销，时间复杂度减少为$O(n^2)$。
- 栈模型与递推算法相似，从前面的项开始计算并记录当前项的数据以用于后面的项的相加计算，从而避免了冗余计算带来的巨大的时间开销。同时栈模型不同于递推算法的后面的项由前面两项相加得到的计算方法，将题目所求方案数简化模拟为出栈顺序数，从递推算法的一步步相加便变为后面的项由前面两项相乘的计算方法，相较于递推算法实现了跳跃计算，时间复杂度仍为$O(n^2)$，但其效率更高，故在购票人数相同的情况下所用时间更短。
- 组合算法将此题运用组合数学的方法简化成一个算数公式，但由于公式内部存在阶乘，需要$O(n)$的时间复杂度计算出数字答案，相较于前四种算法效率最高，耗时最短。

实现

算法函数

算法一 搜索策略

```
long long int cnt;

//bool scheme[200];//true表示50元，false表示100元
int n;//2n表示人数
int k;//剩余50张数
int done;//已购票人数
int res_100, res_50;//剩余手持/100元/50元的人数
void search(int flag) {

    if (res_50 == 1 && flag) {
        /*scheme[done] = true;*/
        ++done;
        --res_50;
        ++k;
        ++cnt;
        /*printf("\b方案%d：\b", cnt);
        for (int i = 0; i < done; ++i) {
            printf("    第%d人：", i + 1);
            scheme[i] ? printf("%d", 50) : printf("%d", 100);
        }
        for (int i = done + 1; i < 2 * b; ++i) {
            printf("    第%d人：", i);
            printf("%d", 100);
        }*/
        return;
    }

    if (!k && !flag) {
        ++done;
        --res_100;
        --k;
        return;
    }
    if (flag) {
        --res_50;
        /*scheme[done] = true;*/
        ++done;
        ++k;
    }
    else {
        --res_100;
        /*scheme[done] = false;*/
        ++done;
        --k;
    }

    if (res_50) {
        search(true);
```

```
        ++res_50;
        --done;
        --k;
    }
    if (res_100) {
        search(false);
        ++res_100;
        --done;
        ++k;
    }
    return;
}
```

算法二 栈模型

```
int n;
long long int cnt;
int s_f[100], sum[100];

void stackmodel() {

    sum[0] = sum[1] = 1;//0个元素、1个元素出栈顺序都是1种
    for (int i = 2; i <= n; ++i) {
        for (int j = 1; j <= i; ++j) {
            s_f[j] = sum[j - 1] * sum[i - j];
            sum[i] += s_f[j];
        }
    }
    cnt = sum[n];
    return;
}
```

算法三 递归算法

```
int recursion(int a, int b) {

    if (a < b)
        return 0;
    if (!b)
        return 1;
    return recursion(a, b - 1) + recursion(a - 1, b);
}
```

算法四 递推算法

```
int r_f[100][100];
void recurrence() {
    for (int i = 0; i <= n; ++i) {
        for (int j = 0; j <= n; ++j) {
            if (!j)
                r_f[i][j] = 1;
            else if (i < j)
                r_f[i][j] = 0;
            else
                r_f[i][j] = r_f[i][j - 1] + r_f[i - 1][j];
        }
    }
    cnt = r_f[n][n];
    return;
}
```

算法五 组合算法

```
void combination() {

    int upnum = 1, downnum = 1;

    int i = n;
    int j = 2 * n;
    for(; i > 0; --i, --j){
        upnum *= j;
        downnum *= i;
    }
    cnt = (upnum/downnum)/(n + 1);
}
```