**华南农业大学**

**《算法分析与设计》课程实验**

**实验4：找零钱**

年级专业： 21信息与计算科学

学生姓名： 吴嘉豪

学生学号： 202125810321

指导老师： 赵峰

实验时间： 2024年 04月 29日

一、实验内容

某国发行种面额的钞票，组成面额集合Denominations。现需要找零钱给顾客，根据面额Denominations与要找的零钱数Change，设计算法给出面额组合，要求使用的钞票张数最少。

要求用动态规划与贪心算法两种方法。

假设:（1）每种钞票都有足够的张数；

（2）零钱数Change与面额均为正整数，且最小面额为1（保证有解）。

二、实验步骤

1、动态规划：确定递归量，找出递归公式，并证明或解释递归公式的合理性；编写动态规划程序；

2、贪心算法：选择合适的贪心策略，证明或解释贪心策略的合理性；编写贪心算法程序；

3、运行测试数据，检查两种算法得到的是否是最优解；

4、分析、比较两种算法的时间复杂度；分析贪心算法在何种情况下能得到最优解；

5、填写实验报告。

三、实验过程与结果

1．

定义一个数组dp ， dp[i]，表示 当前零钱数为i的时候，找的钞票的数量为dp[i]， 那么我们可以很容易得到

* dp[j] = dp[j – Denominations[now]] + 1 (dp[j – Denominations[now]] != -1 && dp[j] == -1 && Denominations[now] >=0)
* dp[j] = min(dp[j], dp[j – Denominations[now]] + 1) (dp[j – Denominations[now]] != -1 && dp[j] != -1 && j – Denominations[now] >=0)

这个两个等式必须满足一个。

我们知道 dp[j - Denominations[now]] + 1 是最优的 找钱数， 那么 dp[j] = dp[j - Denominations[now]] + 1 一定是最优的找钱数（当零钱数为j的时候）

因为不可能存在dp[j] = dp[j - Denominations[now]] + 1的情况

2.

我们这里选择的贪心策略是， 每次， 选择 金额最大的钞票， 然后去抵消零钱数，知道零钱数小于当前这个钞票， 然后找下一个比它小的钞票。

但是这个是存在问题的。 4下面分析

3.

-------前提---------

当前的找钱数量为103

当前的钞票的集合为:

[1, 17, 19, 10, 12, 19, 6]

------- 动态规划 -------

当找钱金额为 103 时最小的找钱数量为6

------- 贪心算法 -------

当前找钱的金额为 103 时最小(这是用这个贪心算法找出来的)的找钱数量为8

很显然 贪心策略不一定奏效

4. 对于动态规划来说,时间复杂度为O(Change \* n)

对于这个伪贪心算法来说，时间复杂度为O(n)

贪心算法只有在 钞票的 集合是 从 1 到 m 所有的数都是存在的时候 才可以成立 （m <= n）

四、源程序代码

#*include*<iostream>

#*include*<vector>

#*include*<algorithm>

#*include*<string>

#*include*<ctime>

#*include* <iomanip>

using namespace std;

void *solve*() {

// *动态规划法*

    int n;

    n = *rand*() % 10 + 5;

    vector<int>*Denominations*(n);

*for*(int i = 0; i < n; ++i){

        Denominations*[*i*]* = *rand*() % 20 + 1;

    }

// *保证答案一定会存在*

    Denominations*[*0*]* = 1;

    int change = *rand*() % 100 + 30;

// *dp[i] = 0 dp[i] = -1;*

// *dp[j] = dp[j - Denominations[now]] + 1*

    vector<int>*dp*(change + 1, -1);

    dp*[*0*]* = 0;

*for*(int i = 1; i <= change; ++i){

*for*(int j = 0; j < n; ++j){

*if*(Denominations*[*j*]* > i) *continue*;

*if*(dp*[*i - Denominations*[*j*]]* != -1) {

*if*(dp*[*i*]* == -1) dp*[*i*]* = dp*[*i - Denominations*[*j*]]* + 1;

*else* dp*[*i*]* = *min*(dp*[*i*]*,  dp*[*i - Denominations*[*j*]]* + 1);

            }

        }

    }

    cout*<<*"-------前提---------"*<<endl*;

    cout*<<*"当前的找钱数量为"*<<*change*<<endl*;

    cout*<<*"当前的钞票的集合为: "*<<endl*;

*for*(int i = 0; i < n; ++i){

*if*(i == 0) cout*<<*"[";

        cout*<<*Denominations*[*i*]*;

*if*(i != n - 1){

            cout*<<*", ";

        }

*else* cout*<<*"]\n"*<<endl*;

    }

    cout*<<*"------- 动态规划 -------"*<<endl*;

    cout*<<*"当找钱金额为 "*<<*change*<<*" 时最小的找钱数量为"*<<*dp*[*change*]<<endl*;

    cout*<<*"------- 贪心算法 -------"*<<endl*;

*sort*(Denominations.*rbegin*(), Denominations.*rend*());

    int billCnt = 0;

    int tempChange =change;

*for*(int i = 0; i < n; ++i){

        int billItemCnt =  (tempChange) / Denominations*[*i*]*;

        billCnt += billItemCnt;

        tempChange -= billItemCnt \* Denominations*[*i*]*;

    }

    cout*<<*"当前找钱的金额为 "*<<*change*<<*" 时最小(这是用这个贪心算法找出来的)的找钱数量为"*<<*billCnt*<<endl*;

}

int *main*() {

    int t;

    t = 1;

*srand*(*time*(nullptr));

    cin*>>*t;

*while*(t--){

*solve*();

    }

*return* 0;

}

五、评语及评分

|  |
| --- |
| 评语及评分  评阅人签名： |