**华南农业大学**

**《算法分析与设计》课程实验**

**实验3：0-1背包问题**

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21日

一、实验内容

用动态规划与分治法解决0-1背包问题，比较并分析试验结果。

二、实验步骤

1、编写动态规划与分治法求解0-1背包问题的程序；

2、用测试样例测试算法的正确性；

3、随机生成5、10、20、30、40个物品的数据（重量与价值），设置适当的背包承重，分别用动态规划与分治法求解，记录运行时间；

4、以物品个数为轴、运行时间为轴，以图形的方式比较运行结果（动态规划的运行时间同一种颜色，再用同种颜色的线顺次连接成曲线；分治法的运行时间用另一种颜色，在用该颜色的线顺次连接成曲线）。

5、分析时间复杂度（分治法的量力分析），填写实验报告。

三、实验结果

3

当前的n为：5

-----01背包算法-----

当背包的容量为: 961

获得的价值为: 1392

消耗的时间为0ms

-----分治法-----

当背包的容量为：961

获得的价值为：1392

消耗的时间为19ms

当前的n为：10

-----01背包算法-----

当背包的容量为: 782

获得的价值为: 1484

消耗的时间为0ms

-----分治法-----

当背包的容量为：782

获得的价值为：1484

消耗的时间为31ms

当前的n为：20

-----01背包算法-----

当背包的容量为: 1099

获得的价值为: 4805

消耗的时间为5ms

-----分治法-----

当背包的容量为：1099

获得的价值为：4805

消耗的时间为111ms

当前的n为：30

-----01背包算法-----

当背包的容量为: 637

获得的价值为: 3175

消耗的时间为1ms

-----分治法-----

当背包的容量为：637

获得的价值为：3175

消耗的时间为53ms

4.

因为背包容量是随机生成的，并且n的数值很小，所以造成n=30的时候耗时比n=20的时候要小，并且分治法需要频繁的创建数组， 也是耗时的重要原因图形用户界面

中度可信度描述已自动生成

5．

对于 01 背包， 每次我是先遍历所有的物品， 然后再遍历从大到小 遍历背包容量所以时间复杂度为O(n \* V) (V 代表背包容量)

对于分治法， 我是再每次的分治 阶段 从 数组a 和 数组b 遍历 选择 加起来最大价值的。 根据分治的特性，那么时间复杂度为O(V^2 \* logn \* n)

四、源程序代码

#*include*<iostream>

#*include*<algorithm>

#*include*<ctime>

#*include* <numeric>

#*include*<vector>

#*include*<string>

#*include* <iomanip>

#*include*<functional>

#*include*<cstdio>

#*include*<cstring>

using namespace std;

int n = 0;

void *solve*() {

    vector<int> *value*(n);

    vector<int> *cost*(n);

    int packSum = *rand*() % 1000 + 500;

// *初始化*

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

        value*[*i*]* = *rand*()%1000 + 2;

        cost*[*i*]*  =  *rand*() % 1000 + 2;

    }

    function<void()> pagAlg = [&](){

// *计算总的 和*

        int dp[packSum + 1];

*memset*(dp, 0, sizeof dp);

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

*for*(int j = packSum; j >= cost*[*i*]*; --j){

                dp[j] = *max*(dp[j], dp[j - cost*[*i*]*] + value*[*i*]*);

            }

        }

        cout*<<*"-----01背包算法-----"*<<endl*;

        cout*<<*"当背包的容量为: "*<<*packSum*<<endl*;

        cout*<<*"获得的价值为: "*<<*dp[packSum]*<<endl*;

    };

    function<*vector*<int>(int, int, vector<int>, vector<int>)> divDo = [&](int l, int r, vector<int> nums1, vector<int> nums2){

        vector<int> *nums*(packSum + 1, 0);

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

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

// *i 和 j 的 价值 为 左边的 dp[i] 和 右边 dp[j] 最优的 进行抉择*

                nums*[*i + j*]* = *max*(nums*[*i + j*]*, nums1*[*i*]* + nums2*[*j*]*);

            }

        }

// *后面的容量可能会劣于 前面的容量*

*for*(int j = 1; j <=packSum; ++j) nums*[*j*]* = *max*(nums*[*j*]*, nums*[*j - 1*]*);

*return* nums;

    };

    function<*vector*<int>(int ,int)> divAlg = [&](int l,int r){

*if*(l > r) {*return* *vector*<int>(packSum + 1, 0);}

*if*(l == r){

            vector<int> *nums*(packSum + 1, 0);

*if*(cost*[*l*]* <= packSum)

            nums*[*cost*[*l*]]* = value*[*l*]*;

*return* nums;

        }

        int mid = (l + r)>>1;

// *分治*

        vector<int> nums1 =  *divAlg(*l, mid*)*;

        vector<int> nums2 = *divAlg(*mid + 1, r*)*;

// *干活*

*return* *divDo(*l, r, nums1, nums2*)*;

    };

    int start = *clock*();

*pagAlg()*;

    int end = *clock*();

    cout*<<*"消耗的时间为"*<<*end - start*<<*"ms"*<<endl*;

    cout*<<endl*;

    start = *clock*();

    vector<int> temp =  *divAlg(*0, n - 1*)*;

    end = *clock*();

    cout*<<*"-----分治法-----"*<<endl*;

    cout*<<*"当背包的容量为："*<<*packSum*<<endl*;

    cout*<<*"获得的价值为："*<<*temp*[*packSum*]<<endl*;

    cout*<<*"消耗的时间为"*<<*end - start*<<*"ms"*<<endl*;

    cout*<<endl*;

}

int *main*() {

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

    int numsN[5] = {5, 10, 20, 30, 40};

    int t = 4;

    int k = 0;

*while*(t--){

        n = numsN[k];

        cout*<<*"当前的n为："*<<*n*<<endl*;

*solve*();

        k++;

    }

*return* 0;

}

五、评语及评分

|  |
| --- |
| 评语及评分  评阅人签名： |