

**《算法分析与技术》实验报告**

学生姓名： 谭哲文（8202191123）

指导老师： 李 敏

2023年06月

**实验五 贪心算法和随机算法**

1. **概述：**

**贪心算法：**

(１) 贪心性质：整体的最优解可通过一系列局部最优解达到，并且每次的选择可以依赖以前做出的选择，但不能依赖于以后的选择。

(２) 最优子结构：问题的整体最优解包含着它的子问题的最优解。

基本步骤：

(１) 分解：将原问题分解为若干相互独立的阶段。

(２) 解决：对于每一个阶段求局部的最优解。

(３) 合并：将各个阶段的解合并为原问题的解。

**实验5.1 背包问题**

**（1）实验内容**

A.问题描述

给定几组数据，利用贪心算法的思想，将物品装入背包并使得其价值最大。

B.实验步骤

① 计算每种物品单位重量的价值 Vi / Wi 。

② 依贪心选择策略，将尽可能多的单位重量价值最高的物品装入背包。

③ 若将这种物品全部装入背包后，背包内的物品总重量未超过 C, 则选择单位重

量价值次高的物品并尽可能多地装入背包。

④ 依此策略一直地进行下去，直到背包装满为止。

**（2）问题描述分析与算法设计思想**

利用贪心算法求解背包问题，输出相应结果，并计算出程序运行所需要的时间。背包

问题简介如下：假设有 n 个物品，每个物品 i 的价值为 vi，⼤⼩为 si。包的容量为 S， 要求从这 n 个物品中挑选若⼲物品装进包中，在所有装进包中物品的⼤⼩于等于包容量 S 的

前提下，包中物品总价值最⼤。

（分数）背包问题与 0-1 背包问题的区别在于，完全背包问题可以只将物品的一部分

装入背包；那么对于背包问题思路很明确，优先将单位重量价值最高的物品放入，最终一

定能得到装满背包所能达到的最大的价值；

首先，给定几种数据。固定几个随机数组；然后，求解背包问题，使背包装满并且使得其价值最大。

贪心算法：

① 计算每种物品单位重量的价值 Vi / Wi 。

存储在数组中，便于调用计算。

② 依贪心选择策略，将尽可能多的单位重量价值最高的物品装入背包。

通过排序算法，将存有物品单位重量的价值的数组进行排序，从值最大的

物品开始选择。

③ 若将这种物品全部装入背包后，背包内的物品总重量未超过 C, 则选择单位重

量价值次高的物品并尽可能多地装入背包。

判断比较此时包的容纳能力和物品重量，选择是否装入物品。

④ 依此策略一直地进行下去，直到背包装满为止。

循环以上三步，直到背包装满。

**（3）代码实现**

对于任一物品，拥有重量、价值（、编号（便于统计）），将其作为一个整体（排序）

① 计算每种物品单位重量的价值 Vi / Wi 。存储在数组中，便于调用计算。

② 依贪心选择策略，将尽可能多的单位重量价值最高的物品装入背包。通过排序

算法，将存有物品单位重量的价值的数组进行排序，从值最大的物品开始选择。

③ 若将这种物品全部装入背包后，背包内的物品总重量未超过 C, 则选择单位重

量价值次高的物品并尽可能多地装入背包。判断比较此时包的容纳能力和物品重量，选

择是否装入物品。

④ 依此策略一直地进行下去，直到背包装满为止。

循环以上三步，直到背包装满。

**算法部分：**

#include <iostream>

using namespace std;

// 按照单位重量的价值量大小降序排列

void Sort(int n, float \*w, float \*v)

{

    int i, j;

    float temp1, temp2;

    for (i = 1; i <= n; i++)

        for (j = 1; j <= n - i; j++) // 冒泡排序

        {

            temp1 = v[j] / w[j];

            temp2 = v[j + 1] / w[j + 1];

            if (temp1 < temp2)

            {

                swap(w[j], w[j + 1]);

                swap(v[j], v[j + 1]);

            }

        }

}

int main()

{

    float w[101]; // 用来表示每个物品的重量

    float v[101]; // 用来表示每个物品的价值量

    float x[101]; // 表示最后放入背包的比例

    int n;        // 物品数

    float M;      // 背包最大容纳重量

    cout << "请输入物品数：" << endl;

    cin >> n;

    cout << "请输入最大容量：" << endl;

    cin >> M;

    // 依次输入每件物品的重量和价值量

    cout << "请依次输入每件物品的重量和价值量：" << endl;

    for (int i = 1; i <= n; i++)

        cin >> w[i] >> v[i];

    // 按照单位重量的价值量大小降序排列

    Sort(n, w, v);

    int i;

    for (i = 1; i <= n; i++)

        x[i] = 0; // 初始值，未装入背包，x[i]=0

    float c = M;  // 更新背包容纳量

    for (i = 1; i <= n; i++)

    {

        if (c < w[i])

            break; // 不能完全装下

        x[i] = 1;

        c = c - w[i];

    }

    if (i <= n)

        x[i] = c / w[i];

    // 输出

    double sum = 0;

    for (int i = 1; i <= n; i++)

    {

        cout << "重量为" << w[i] << "价值量为" << v[i] << "的物品"

             << "放入的比例为" << x[i] << endl;

        sum += x[i] \* v[i];

    }

    cout << "总价值为：" << endl

         << sum<<endl;

    system("pause");

    return 0;

}

**运行结果：**

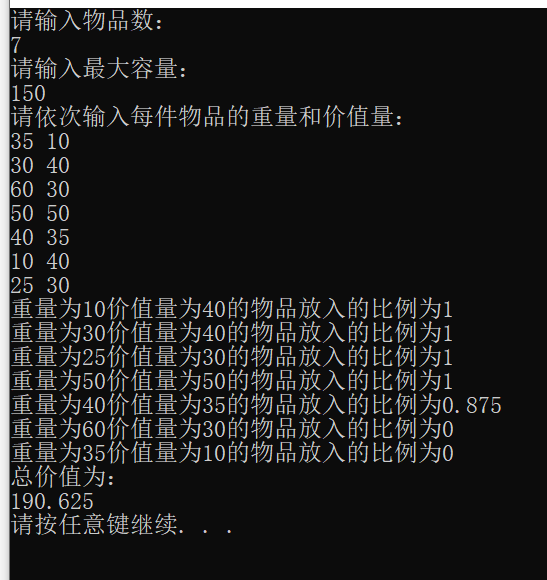
7  
150  
35 10  
30 40  
60 30

50 50

40 35

10 40

25 30



1. **算法及问题分析**

一般来说，贪心算法的证明围绕着：整个问题的最优解一定由在贪心策略中存在的子问题的最优解得来的。

对于背包问题中的3种贪心策略，都是无法成立(无法被证明)的，解释如下：

贪心策略：选取价值最大者。反例：

W=30

物品：A B C

重量：28 12 12

价值：30 20 20

根据策略，首先选取物品A，接下来就无法再选取了，可是，选取B、C则更好。

(2)贪心策略：选取重量最小。它的反例与第一种策略的反例差不多。

(3)贪心策略：选取单位重量价值最大的物品。反例：

W=30

物品：A B C

重量：28 20 10

价值：28 20 10

根据策略，三种物品单位重量价值一样，程序无法依据现有策略作出判断，如果选择A，则答案错误。

搬桌子问题

**（1）实验内容**

某教学大楼一层有n个教室，从左到右依次编号为1、2、…、n。现在要把一些课桌从某些教室搬到另外一些教室，每张桌子都是从编号较小的教室搬到编号较大的教室，每一趟，都是从左到右走，搬完一张课桌后，可以继续从当前位置或往右走搬另一张桌子。输入数据：先输入n、m，然后紧接着m行输入这m张要搬课桌的起始教室和目标教室。输出数据：最少需要跑几趟。

**Sample Input**

10 5

1 3

3 9

4 6

6 10

7 8

**Sample Output**

3

分析：贪心算法，把课桌按起点从小到大排序，每次都是搬离当前位置最近的课桌。

1. **代码分析**

#include <stdio.h>

#include<iostream>

using namespace std;

int main()

{

    struct

    {

        int start;

        int end;

    } a[100];

    int i, j;

    int n, m, min, num, temp, used[100] = {0};

    cout<<"Sample Input:\n";

    cin>>m>>n;

    for (i = 0; i < n; i++)

        cin>>a[i].start>>a[i].end;

    min = 0;

    num = 0;

    while (num < n)

    {

        temp = 0;

        for (i = 0; i < n; i++)

            if (used[i] == 0 && a[i].start >= temp)

            {

                temp = a[i].end;

                used[i] = 1;

                num++;

            }

        min++;

    }

    cout<<"Sample Out:\n";

    cout<<min<<endl;

    system("pause");

}

**运行结果：**

