**大 连 海 事 大 学**

**\_\_\_\_\_\_\_\_\_\_\_\_ 实 验 报 告**

**算法分析与设计**

|  |
| --- |
| 专 业 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 姓名\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 学号\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **电子信息**  **1120211433**  **杨显鹏**  实验日期\_\_\_\_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_**\_\_ 指导教师\_\_**\_\_\_\_\_\_\_\_\_\_\_**\_\_\_\_\_\_\_  **2021.11.06**  **曲衍鹏** |
| 第\_\_\_\_\_\_\_\_实验 实验名称 \_\_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**\_\_\_\_\_\_\_\_\_  **采用动态规划方法实现0-1背包问题**  **三次** |

**实验三：采用动态规划方法实现0-1背包问题**

**一、实验目的**

理解动态规划方法的核心思想以及动态规划方法的求解过程；从算法分析与设计的角度，对0-1背包问题的基于DP法求解有更进一步的理解。

**二、实验内容**

(1) 编程实现采用动态规划方法求解0-1背包问题的算法。

(2) 问题描述：给定n种物品和一个容量为M的背包，第i个物品的重量为wi，效益值为pi，假设该问题的解向量为(x1, x2, … , xn)，其中，若xi=1表示第i个物品放在包中；若xi=0表示第i个物品没有被选中，i=1, 2, … , n。要求依据动态规划思想构造一个装包算法，使得背包的总效益值达到最大，并加以实现。

极大化函数： 达到极大。

约束条件： ≤ M。xi=0 或xi=1。

(3) 假设n=20，M小于20个物品的总重量。随机生成(wi, pi)，每个wi和pi均为正整数，i=1, 2, … , n。输入数据用文件保存，以便打印输出。

(4) 输出生成的输入数据及根据输入数据所得到的解，并说明所得到的解是否为最优解。

(5) 根据实验结果，撰写实验报告。

**三、实验数据**

设定背包的容量为10，共有物品5个，其重量和物品价值如表1所示：

表1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 物品编号 | 1 | 2 | 3 | 4 | 5 |
| 物品重量 | 2 | 3 | 4 | 5 | 6 |
| 物品价值 | 6 | 5 | 4 | 3 | 2 |

**四、程序代码**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<time.h>

typedef struct coordinate{

float x;

float y;

}Point;

float max(float a,float b){

return a>b?a:b;

}

void rands(Point xy[],int n){

int i;

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

xy[i].x=rand()/(RAND\_MAX+1.0)\*100;

xy[i].y=rand()/(RAND\_MAX+1.0)\*100;

}

return;

}

void paixux(Point a[],int low,int high){

Point tmp;

tmp=a[low];

int i=low;

int j=high;

if(low<high){

while(i<j){

while(i<j&&a[j].x>tmp.x){

j--;

}

a[i]=a[j];

while(i<j&&a[i].x<tmp.x){

i++;

}

a[j]=a[i];

}

a[i]=tmp;

paixux(a,low,i-1);

paixux(a,i+1,high);

}

else{

return;}

}

float mins(Point xy[],int n){

int i,j;

float sum=0;

float s[4];

rands(xy,n);

paixux(xy,0,n-1);

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

if(i==0){

s[0]=xy[i].x\*xy[i].y;

s[1]=xy[i].x\*(100-xy[i].y);

sum+=max(s[0],s[1]);

}

else if(i==n-1){

s[0]=(xy[i].x-xy[i-1].x)\*xy[i].y;

s[1]=(xy[i].x-xy[i-1].x)\*(100-xy[i].y);

s[2]=(100-xy[i].x)\*xy[i].y;

s[3]=(100-xy[i].x)\*(100-xy[i].y);

sum+=max(s[0],max(s[1],max(s[2],s[3])));

}

else{

s[0]=(xy[i].x-xy[i-1].x)\*xy[i].y;

s[1]=(xy[i].x-xy[i-1].x)\*(100-xy[i].y);

sum+=max(s[0],s[1]);

}

}

return sum;

}

int main(){

int n[]={5,10,13,3,15,16,20,25,26,27};

float m;

for(int i=0;i<=9;i++){

Point xy[101];

m=mins(xy,n[i]);

printf("点的个数为%d",n[i]);

printf("最大矩形面积之和：%f\n",m);

printf("最大矩形面积之和占总面积的比例：%f\n",m/10000);

}

printf("over");

}

**五、总结与体会**

**5.1实验结果**

最终实验结果如图1所示，根据物品的重量和物品的价值，并结合背包的容量计算得出最大的利益。

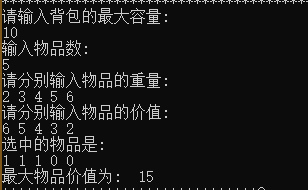


图1

**5.2分析**

每种物品仅有一件，可以选择放或不放。

用子问题定义状态：即f[i][v]表示前i件物品恰放入一个容量为v的背包可以获得的最大价值。则其状态转移方程便是：

f[i][v]=max{f[i-1][v],f[i-1][v-c[i]]+w[i]}

动态规划：动态规划是求解决策过程最优化的数学方法。如果一个问题可以分解成若干个子问题，并且子问题之间还有重叠的更小的子问题，就可以考虑用动态规划来解决这个问题。

应用动态规划之前要分析能否把大问题分解成小问题，分解后的每个小问题也存在最优解。如果将小问题的最优解组合起来能够得到整个问题的最优解，那么就可以使用动态规划解决问题。

可以应用动态规划求解的问题主要由四个特点：

1. 问题是求最优解

2. 整体问题的最优解依赖于各个子问题的最优解

3. 大问题分解成若干小问题，这些小问题之间还有相互重叠的更小的子问题

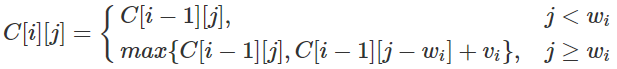
4. 从上往下分析问题，从下往上求解问题

实验步骤：

算法设计，包括策略与数据结构的选择，设计用二维数组对物品信息进行记录： C[i][j]用来记录如果当前还有i个物品，背包容量还剩j的情况下，当前背包所能得到的最大价值。很容易发现条件即：

1

递归定义应该为：



可以这样理解，每个物品我可以选择是否加入到背包中，首先判断，当前物品是否重量已经大于背包所能容纳的重量；如果能容纳该物体，则进行判断加入该物品 得到的价值更高，还是不加入该物品 所能得到的物品的总价值更高。

**5.3总结**

动态规划算法通常是用来解决某种最优性质的问题。基本思想是将带求解问题划分为若干个子问题，先求解子问题，然后从子问题的解得到原问题的解。动态规划与分治法的区别在于，动态规划的子问题可能是互相重叠的重复计算的，分治法则是相互独立的。可以用一个表来记录子问题是否已经求解，这样可以避免重复求解。

需要满足最优化原理、无后效性和重叠性。最优化原理，一个最优化策略的子策略一定是最优的，就是满足最优子结构的性质；无后效性，一个阶段以前各阶段的状态无法直接硬性它未来的决策，只能通过当前的这个状态；重叠性，就是记录已经解决过的问题，需要存储已经解决过的问题，空间复杂度比较大，是一种以空间换时间的算法；

通过本次课程设计，我对动态规划算法有了初步了解有了更加深刻的了解，更巩固了课堂中有关于动态算法求解0-1背包问题的知识，真正的学会了一种算法。