**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 \_\_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**\_\_\_\_\_\_\_\_\_  **正方形切割问题**  **二次** |

**实验二：正方形切割问题**

**一、实验目的**

通过正方形分割对贪婪算法有进一步了解，学会分析问题，提高解决问题的能力。

**二、实验内容**

(1) 编程实现求解正方形切割问题的算法，并输出切割部分的面积。要求切割时，采用贪婪算法的思想，每次选择面积尽可能大的长方形进行切割。

(2) 问题描述：给定一个100\*100的正方形A，假设将A的左上角顶点视为原点，并定义其坐标为(0, 0)。在A中自动生成不超过100个互不相同的点(为简单起见，可假设每个点的横坐标和纵坐标各不相同，这样可保证每条横线或竖线上至多只有一个点，但该约定不是必须的)，且必有一个点恰好在原点上，要求依据这些给定的点切割正方形，切割方向只能向下和向右，每次都寻找最大的长方形进行切割，但所切割的长方形内部不能含有任何其它给定点(给定点可在切割线上)。已经切割的部分不可重复切割，且切割出的部分必须是长方形。对每个给定点都必须做切割操作，并累计切割出的面积，使切割出的总面积尽可能大。

(3) 将原点作为一个给定点，随机生成其余99个点，并使得这些点的横坐标和纵坐标各不相同。

(4) 要求至少随机构造10组数据(点)，输出每组数据的运行结果。若可能(选择做)，将切割过程可视化。

(5) 求每组数据的切割总面积与正方形面积的比，给出相应的结论。

(6) 根据实验结果，撰写实验报告。

**三、实验数据**

实验数据随机选择10个整数如表1所示：

表1

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数据编号 | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
| 数据值 | 5 | | 10 | 13 | 3 | 15 | 16 | 20 | 25 | 26 | 27 |

**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，采用随机选取的10个点数，计算出其点数、最大面积之和与最大矩形面积占总面积的比例。

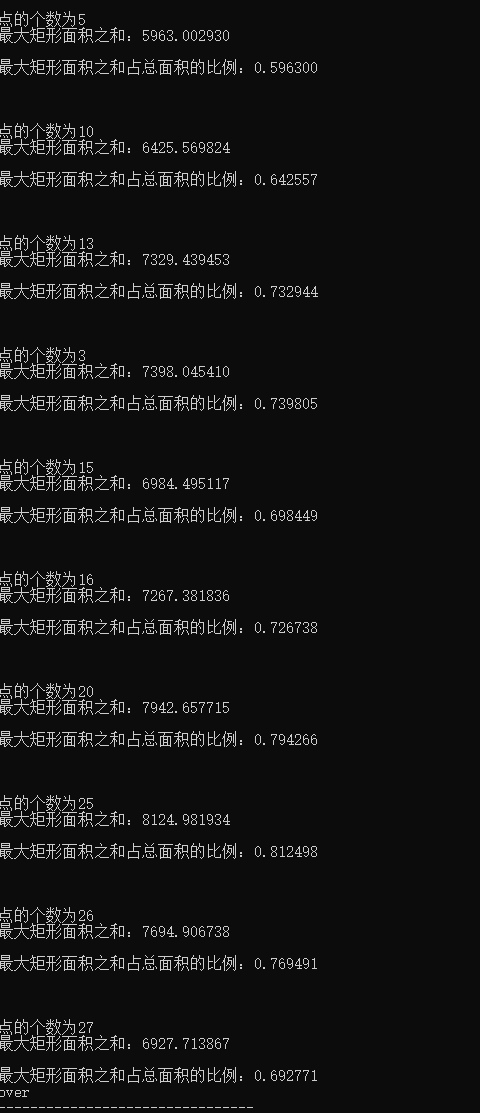


图1

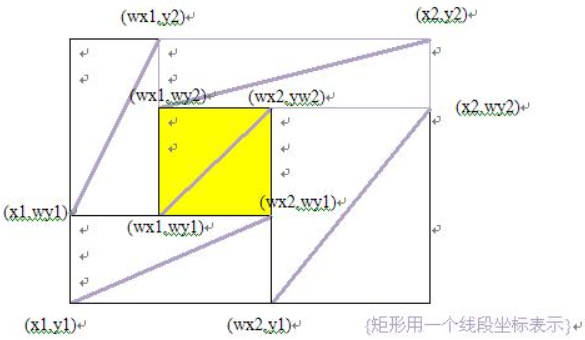
**5.3分析**

贪婪算法：贪婪算法可解决的问题通常大部分都有如下的特性：

随着算法的进行，将积累起其它两个集合：一个包含已经被考虑过并被选出的候选对象，另一个包含已经被考虑过但被丢弃的候选对象。有一个函数来检查一个候选对象的集合是否提供了问题的解答。该函数不考虑此时的解决方法是否最优。还有一个函数检查是否一个候选对象的集合是可行的，也即是否可能往该集合上添加更多的候选对象以获得一个解。和上一个函数一样，此时不考虑解决方法的最优性。选择函数可以指出哪一个剩余的候选对象最有希望构成问题的解。最后，目标函数给出解的值。

为了解决问题，需要寻找一个构成解的候选对象集合，它可以优化目标函数，贪婪算法一步一步的进行。起初，算法选出的候选对象的集合为空。接下来的每一步中，根据选择函数，算法从剩余候选对象中选出最有希望构成解的对象。如果集合中加上该对象后不可行，那么该对象就被丢弃并不再考虑；否则就加到集合里。每一次都扩充集合，并检查该集合是否构成解。如果贪婪算法正确工作，那么找到的第一个解通常是最优的。

对于正方形切割问题：对于求解若干个矩形的面积的交集，往往采用矩形切割思想，简单来说拿到若干个矩形，求任意一个矩形不被覆盖的面积总和。



我们假设有2个矩形：A和B，A用白色染色，B用黄色染色，A放在底层，B放在A之上，这样，黄色区域就是矩形A和矩形B的交集了，我们可以发现

wx1=max(wx1,x1)

wx2=min(wx2,x2)

wy1=max(wy1,y1)

wy2=min(wy2,y2)

这样，我们判断是否存在，不存在则没有，结束，否则继续上浮，寻找下一个重叠矩阵

**5.2总结**

贪心算法（又称贪婪算法）在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的是在某种意义上的局部最优解。

贪心算法不是对所有问题都能得到整体最优解，关键是贪心策略的选择，选择的贪心策略必须具备无后效性，即某个状态以前的过程不会影响以后的状态，只与当前状态有关。

通过本次课程设计，我对贪婪算法有了更加深刻的了解，更巩固了课堂中有关于贪婪算法的知识，真正的学会了一种算法。