**算法分析与设计实验报告**

**正方形切割问题**

姓 名： 于 欣 悦

专 业： 软件工程

学 号： 1120191350

实 验 日 期：2019年12月12日

1. **实验目的**

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。

1. **实验内容**

通过贪婪法，在切割区域，随机生成n个点，然后遍历以各个点向下向右方向的切割，每一次的切割面积最大。

（1）贪心法的基本思路：

从问题的某一个初始解出发逐步逼近给定的目标，以尽可能快的地求得更好的解。当达到某算法中的某一步不能再继续前进时，算法停止。

（2）实现该算法的过程：

从问题的某一初始解出发； 遍历每个点，计算最大面积，求出可行解的一个解元素； 通过n次的遍历，由所有解元素组合成问题的一个最终解。

1. **实验数据**

表3.1 随切割点数不同比值的变化

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 切割点数 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 5 | 15 | 25 |
| 比值 | 0.5304 | 0.5364 | 0.5116 | 0.5783 | 0.5896 | 0.566 | 0.5703 | 0.5438 | 0.54 | 0.5254 |

1. **程序代码**

#include "stdafx.h"

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include <string.h>

#define TRUE 1

#define FALSE 0

#define MatrixLarge 100

int CutPoint = 0;

int Matrix[MatrixLarge][MatrixLarge];

int CutSize = 0;

int CX[100]; //点的横坐标

int CY[100]; //点的纵坐标

//矩阵初始化

void Init()

{

int i,j;

for(i=0;i<MatrixLarge;i++)

for(j=0;j<MatrixLarge;j++)

Matrix[i][j] = 0;

for(int i = 0; i< 100 ; i++)

{

CX[i] = 0;

CX[j] = 0;

}

}

//输入点数个数

void CutPointSize()

{

int i,x,y;

srand((int) time(NULL));

printf("输入要分割的点数:");

scanf("%d",&CutSize);

Matrix[0][0] = 1;

CX[0] = 0;

CY[0] = 0;

if(CutSize>1)

{

for(i=1;i<CutSize;i++)

{

x = rand()%100;

CX[i] = x;

y = rand()%100;

CY[i] = y;

Matrix[x][y] = 1;

}

}

}

//输出矩阵

void PrintMatrix()

{

int i,j;

for (i=0;i<MatrixLarge;i++)

{

for (j=0;j<MatrixLarge;j++)

{

printf("%3d",Matrix[i][j]);

}

printf("\n");

}

}

//切割的面积

void Cut\_Size()

{

// printf("00000000000000000000\n");

int CutSum = 0,AreaSum = 0;

int MaxArea = 0; //记录最大切割面积

int MyArea[100][100] = {0}; // 保存所有可能切割的面积

int CutMatrix[100][100]= {0}; //保存最终切割结果

int WorkMatrix[100][100] = {0}; //覆盖的面积

int intColor = 0;

int MyColor[4] = {1,2,3,4};

int i, j, k, l, m, x1, y1, p =0;

int xMin, yMin, xMax,yMax;

bool MyFlag;

// printf("00000000000000000001\n");

for(i = 0;i<MatrixLarge;i++)

{

for (j = 0;j<MatrixLarge;j++)

{

CutMatrix[i][j] = 0;

MyArea[i][j] = 0;

WorkMatrix[i][j] = 0;

}

}

// printf("00000000000000000002\n");

MyFlag = TRUE;

intColor = 0;

AreaSum = 0;

// printf("00000000000000000003\n");

do {

// printf("00000000000000000004\n");

MaxArea = 0;

for (i = 1;i<=CutSize;i++)

{

for (k = 1;k<MatrixLarge;k++)

{

for (j=1;j<MatrixLarge;j++)

{

WorkMatrix[i][j] = CutMatrix[i][j];

}

}

// printf("00000000000000000005\n");

for (m = 1;m<=CutSize;m++)

{//printf("00000000000000000006\n");

if((CX[m] == CX[i] &&CY[m] == CY[i]) ||

(CX[m]>CX[i] && CY[m]>CY[i]))

{//printf("00000000000000000007\n");

for (k=CY[m];k<MatrixLarge;k++)

{

for (j = CX[m];j<MatrixLarge;j++)

{

WorkMatrix[j][k] = WorkMatrix[j][k]+1;

}

}

}

}

l = 0;

m = MatrixLarge;

x1 = MatrixLarge;

for (k = CY[i];k<MatrixLarge;k++)

{

for (j = CX[i]; j<x1 ; j++)

{

if (WorkMatrix[j][k]>1)

{//printf("00000000000000000008\n");

l = l+1;

if (k == CY[i])

{//printf("00000000000000000009\n");

MyArea[i][l] = j - CX[i];

y1 = k;

p = j -1;

}

else

{//printf("000000000000000000010\n");

MyArea[i][l] = (x1 - CX[i] + 1)\*(k - CY[i]);

y1 = k -1;

p = x1;

}

if (MyArea[i][l] > MaxArea)

{//printf("00000000000000000011\n");

MaxArea = MyArea[i][l];

xMin = CX[i];

yMin = CY[i];

xMax = p;

yMax = y1;

}

m = j-1;

j = x1;

//printf("00000000000000000012\n");

}

}

//printf("00000000000000000013\n");

x1 = m;

}

if (WorkMatrix[CX[i]][CY[i]] == 1)

{//printf("00000000000000000014\n");

l = l+1;

MyArea[i][l] = (x1 - CX[i] +1)\*(k - CY[i]); //含横坐标相同的点，该点被纵坐标小的点覆

if (MyArea[i][l] > MaxArea)

{

MaxArea = MyArea[i][l];

xMin = CX[i];

yMin = CY[i];

xMax = x1;

yMax = k -1;

}

}

}

if (MaxArea >0)

{

intColor = (intColor + 1)%4;

AreaSum = AreaSum +MaxArea;

for (k = yMin;k<=yMax;k++)

{

for (j = xMin;j<=xMax;j++)

{

CutMatrix[j][k] = 1;

// Matrix[k][j] = MyColor[intColor];

}

}

MyFlag = FALSE;

}

} while(MyFlag == TRUE);

CutSum = 0;

for (k = 1;k<=MatrixLarge;k++)

{

for (j = 1;j<=MatrixLarge;j++)

{

if (CutMatrix[j][k] == 0)

{

CutSum = CutSum +1;

}

}

}

for (i = 0;i<MatrixLarge;i++)

{

for (j = 0;j<MatrixLarge;j++)

{

printf("%3d",WorkMatrix[i][j]);

}

printf("\n");

}

float f = (float)(AreaSum / 400.0);

printf("切割的面积:%4d (400)\n",(int)(400\*f));

printf("切割面积比例:%.4f\n\n",f);

}

void Choice\_Funciton()

{

printf(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 矩形切割问题 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

CutPointSize();

printf("切割点:\n");

PrintMatrix();

printf("切割面积:\n");

Cut\_Size();

Init();

}

int main()

{

int t = 1;

while(t)

{

Choice\_Funciton();

getchar();

//system("PAUSE");

//system("CLS");

//t--;

}

return 0;

}

1. **实验结果**
2. 程序结果输出图如图5.1所示：

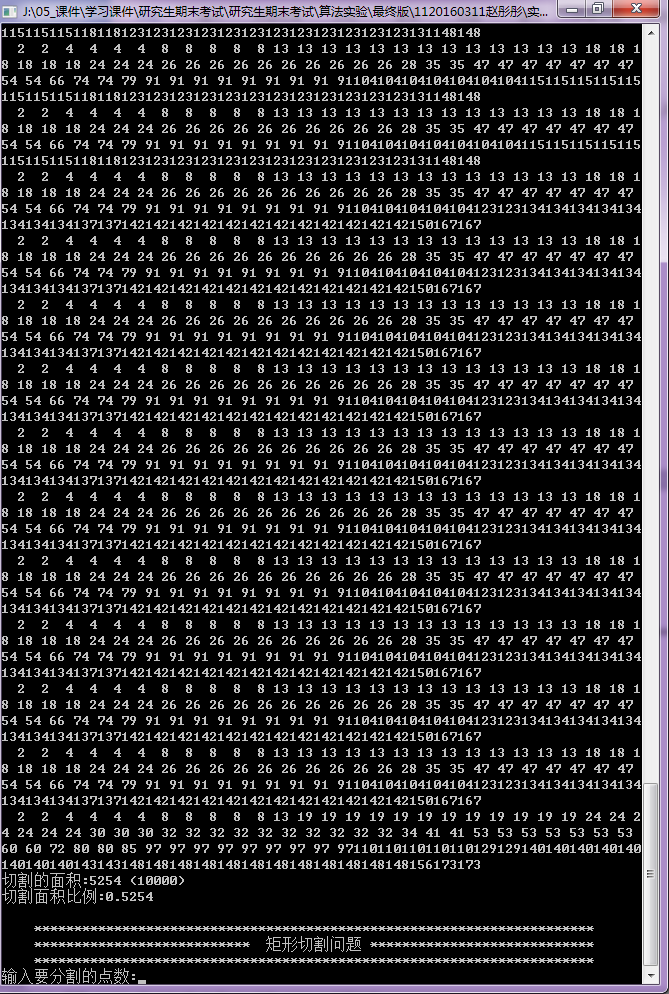


图5.1程序结果输出图

1. 切割总面积与正方形面积的比值散点图如图5.2所示：

图5.2 切割总面积与正方形面积的比值散点图

1. **实验分析及总结**

由切割总面积与正方形面积的比值散点图可以得出要分割的点数越大，分割总面积站正方形面积比例越大。

矩形切割这个问题中来看，计算机可以很容易解决我们一些猜想的问题，但是我们很难用大量的数据和理论去证明。通过此次的实验，了解了这个算法的描述过程，也认识到了写程序的重要性。