学号：5120809022

姓名：黄志强

代码：复制vs，可直接运行

#include <iostream>

#include <vector>

using namespace std;

//结构元素的表示：二维数组（包括指针，行，列，即ptr，row，col）

// 原点：originX 行的index， originY列的index ptr[originX][originY]即是原点所在

//总的来说，是以二位数组存structure element

struct Element{

int\*\* ptr;

int row;

int col;

int originX;

int originY;

Element() :ptr(NULL), row(0), col(0), originX(-1), originY(-1){}

Element(int width, int height)

{

ptr = new int\*[width];

row = width;

col = height;

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

ptr[i] = new int[col];

}

};

//图像的表示方法：二位数组（包括指针，行，列，即ptr，row，col）

//其他的是对二值图像的操作方法

//二位数组取值为0，1

//0代表改点没有图像，只有背景

//1代表改点有图像

class BinaryImage{

public:

BinaryImage()

{

ptr = NULL;

row = 0;

col = 0;

}

BinaryImage( int width, int height) //构造函数

{

row = width;

col = height;

ptr = new int\*[row];

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

ptr[i] = new int[col];

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

{

for (int j = 0; j < col; j++)

ptr[i][j] = 0;

}

}

void setValue(int\*\* p, int width, int height) //赋值函数，可以通过二维数组赋值

{

if (p == NULL || width != row || height != col)

return;

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

{

for (int j = 0; j < col; j++)

ptr[i][j] = p[i][j];

}

}

void setIndexValue(int i, int j, int val) //通过每项来赋值

{

if (i < 0 || j < 0 || i >= row || j >= col)

return;

ptr[i][j] = val;

}

int getIndexValue(int i, int j) //获取值

{

return ptr[i][j];

}

int getRow() //获取行

{

return row;

}

int getCol() //获取列

{

return col;

}

BinaryImage\* dilation(Element& strucEle) //二值图像的dilation操作

{

if (ptr == NULL || row == 0 || col == 0)

return NULL;

int el\_col = strucEle.col - (strucEle.originY+1); //左边应该扩展多少列

int er\_col = strucEle.originY; //右边应该扩展多少列

int et\_row = strucEle.row - (strucEle.originX+1); //上面应该扩展多少行

int ed\_row = strucEle.originX; //下面应该扩展多少行

int d\_row = row + et\_row + ed\_row;

int d\_col = col + el\_col + er\_col;

BinaryImage\* d\_BI=new BinaryImage(d\_row,d\_col);

vector<int>BI\_rows; //原图像有1的位置

vector<int>BI\_cols;

vector<int>SE\_rows; //结构元素有1的位置

vector<int>SE\_cols;

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

{

for (int j = 0; j < col; j++)

{

if (ptr[i][j] == 1)

{

BI\_rows.push\_back(i);

BI\_cols.push\_back(j);

}

}

}

for (int i = 0; i < strucEle.row; i++)

{

for (int j = 0; j < strucEle.col; j++)

{

if (strucEle.ptr[i][j] == 1)

{

SE\_rows.push\_back(i);

SE\_cols.push\_back(j);

}

}

}

for (int i = 0; i < d\_row; i++)

{

for (int j = 0; j < d\_col; j++)

{

//判断是否有交点

if (isDilation(i, j, strucEle.originX, strucEle.originY, SE\_rows, SE\_cols, BI\_rows, BI\_cols, et\_row, el\_col) == true)

d\_BI->setIndexValue(i, j, 1);

}

}

return d\_BI;

}

BinaryImage\* erosion(Element& strucEle)

{

if (ptr == NULL || row == 0 || col == 0)

return NULL;

int e\_row = row; //因为腐蚀图像不可能比原图像大，所以不扩展行和列

int e\_col = col;

BinaryImage\* e\_BI = new BinaryImage(e\_row, e\_col);

vector<int>BI\_rows;

vector<int>BI\_cols;

vector<int>SE\_rows;

vector<int>SE\_cols;

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

{

for (int j = 0; j < col; j++)

{

if (ptr[i][j] == 1)

{

BI\_rows.push\_back(i);

BI\_cols.push\_back(j);

}

}

}

for (int i = 0; i < strucEle.row; i++)

{

for (int j = 0; j < strucEle.col; j++)

{

if (strucEle.ptr[i][j] == 1)

{

SE\_rows.push\_back(i);

SE\_cols.push\_back(j);

}

}

}

for (int i = 0; i < e\_row; i++)

{

for (int j = 0; j < e\_col; j++)

{

//判断结构元素是否包含在原图像内部

if (isErosion(i, j, strucEle.originX, strucEle.originY, SE\_rows, SE\_cols, BI\_rows, BI\_cols) == true)

e\_BI->setIndexValue(i, j, 1);

}

}

return e\_BI;

}

private:

int row; //二维数组表示

int col;

int\*\* ptr;

bool isDilation(int posX, int posY, int originX,int originY, vector<int>&SE\_rows,

vector<int>&SE\_cols, vector<int>&BI\_rows, vector<int>&BI\_cols,int addX,int addY)

{

for (int i = 0; i < SE\_rows.size(); i++)

{

int x=SE\_rows[i] - originX + posX-addX; //结构元素在原图像中对应的x

int y = SE\_cols[i] - originY + posY-addY; //结构元素在原图像中对应的y

for (int j = 0; j < BI\_rows.size(); j++)

{

if (BI\_rows[j] == x&&BI\_cols[j] == y) //判断原图像与结构元素是否相交

return true;

}

}

return false;

}

bool isErosion(int posX, int posY, int originX, int originY, vector<int>&SE\_rows,

vector<int>&SE\_cols, vector<int>&BI\_rows, vector<int>&BI\_cols)

{

for (int i = 0; i < SE\_rows.size(); i++)

{

bool found = false;

int x = SE\_rows[i] - originX + posX; //结构元素在原图像中对应的x

int y = SE\_cols[i] - originY + posY; //结构元素在原图像中对应的y

for (int j = 0; j < BI\_rows.size(); j++)

{

if (BI\_rows[j] == x&&BI\_cols[j] == y) //判断原图像与结构元素是否相交

{

found = true;

break;

}

}

if (found == false) //只要有一个结构元素的1的位置在原图像中找不到，即不包含，那么该点

return false; //不能做erosion操作，即为0

}

return true;

}

};

//简单的测试程序，运行ok，结果ok

int main()

{

BinaryImage a(4, 5); //新建一张图片

a.setIndexValue(0, 0, 1); a.setIndexValue(0, 1, 0); a.setIndexValue(0, 2, 0); a.setIndexValue(0, 3, 1); a.setIndexValue(0, 4, 0);

a.setIndexValue(1, 0, 1); a.setIndexValue(1, 1, 1); a.setIndexValue(1, 2, 1); a.setIndexValue(1, 3, 0); a.setIndexValue(1, 4, 0);

a.setIndexValue(2, 0, 1); a.setIndexValue(2, 1, 0); a.setIndexValue(2, 2, 1); a.setIndexValue(2, 3, 1); a.setIndexValue(2, 4, 0);

a.setIndexValue(3, 0, 1); a.setIndexValue(3, 1, 0); a.setIndexValue(3, 2, 0); a.setIndexValue(3, 3, 0); a.setIndexValue(3, 4, 1);

Element b(2, 2); //新建一个结构元素

b.ptr[0][0] = 1; b.ptr[0][1] = 1;

b.ptr[1][0] = 1; b.ptr[1][1] = 0;

b.originX = 1;

b.originY = 1;

BinaryImage\* k=a.dilation(b); //dilation

for (int i = 0; i < k->getRow(); i++)

{

for (int j = 0; j < k->getCol(); j++)

cout << k->getIndexValue(i, j) << " ";

cout << endl;

}

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

BinaryImage\* l = a.erosion(b); //erosion

for (int i = 0; i < l->getRow(); i++)

{

for (int j = 0; j < l->getCol(); j++)

cout <<l->getIndexValue(i, j) << " ";

cout << endl;

}

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

system("pause");

}