**要求代码和实验报告规范，在算法思想中：对实验涉及的数据结构进行有效设计和分析；对算法进行分析并给出时间、空间复杂度的结论；清晰表达实验思路、出现的问题及解决方法。**

一、调试成功程序及说明

1、

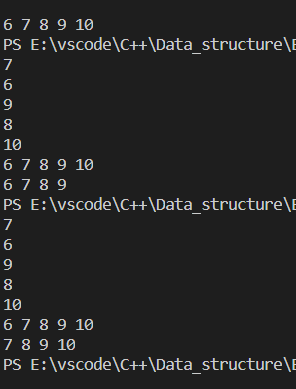
题目：实现二叉排序树的插入和删除。

算法思想：

（1）插入的思想比较简单，所有插入的结点都为叶子结点，也就是说插入一个数据，从跟结点开始向左右结点进行比较，如果小于就往左，大于就往右。

（2）删除的思想略微复杂，分为以下三种情况。①删除叶子结点，直接进行删除②有一棵子树的情况，直接另当前的树指针指向子树，释放这个结点。③具有两棵子树的情况，把右子树最左边的结点与当前的结点的值进行交换，然后删除右子树最左边的结点。

运行结果：



结果分析：运行结果正确，能够正确执行插入和删除操作。

附源程序。

#include <iostream>

#include <cstdlib>

#include <malloc.h>

using namespace std;

#define ElemType int

typedef struct node

{

ElemType data;

struct node \*lchild;

struct node \*rchild;

}\*BinarySortTree, BSNode;

BinarySortTree FindMin(BinarySortTree &BST)

{

if(BST == NULL)

{

return NULL;

}

BinarySortTree cur = BST;

while(cur != NULL)

{

if(cur->lchild ==NULL)

{

return cur;

}

else

{

cur = cur->lchild;

}

}

}

//寻找最小值，返回指针

BinarySortTree InsertBSTree(BinarySortTree &BST, ElemType data)

{

if(BST == NULL)

{

BST = new BSNode;

BST->data = data;

BST->lchild = NULL;

BST->rchild = NULL;

cout << BST->data<<endl;

}

else

{

if(data < BST->data)

{

BST->lchild = InsertBSTree(BST->lchild,data);

}

else if(data > BST->data)

{

BST->rchild = InsertBSTree(BST->rchild,data);

}

}

return BST;

}

//插入一个新的，值为data的结点

BinarySortTree DeleteNode(BinarySortTree &BST, ElemType data)

{

if(BST == NULL)

{

return BST;

}

else if(data < BST->data)

{

BST->lchild = DeleteNode(BST->lchild,data);

}

else if(data > BST->data)

{

BST->rchild = DeleteNode(BST->rchild,data);

}

else

{

if(BST->lchild != NULL && BST->rchild != NULL)

{

BinarySortTree tmp = FindMin(BST->rchild); //找到右子树的最小值，用来代替根节点

BST->data = tmp->data;

BST->rchild = DeleteNode(BST->rchild,tmp->data);

}

else

{

BinarySortTree tmp = BST;

if(tmp->rchild == NULL)

{

BST = BST->lchild;

}

else if(tmp->rchild == NULL)

{

BST = BST->rchild;

}

delete tmp;

}

}

return BST;

}

void InOrderTraverse(BinarySortTree &BST)

{

if(BST == NULL)

{

return ;

}

InOrderTraverse(BST->lchild);

cout << BST->data<<" ";

InOrderTraverse(BST->rchild);

return;

}

int main()

{

BinarySortTree BST = NULL,temp = NULL;

int a[30] = {-1,7,6,9,8,10,5,2,0,10,11};

for(int i = 1; i <= 5; i++)

{

InsertBSTree(BST,a[i]);

}

InOrderTraverse(BST);

int t = 6;

temp=FindMin(BST);

DeleteNode(BST,t);

cout << endl;

//cout << temp->data<<endl;

InOrderTraverse(BST);

return 0;

}

2、

题目：

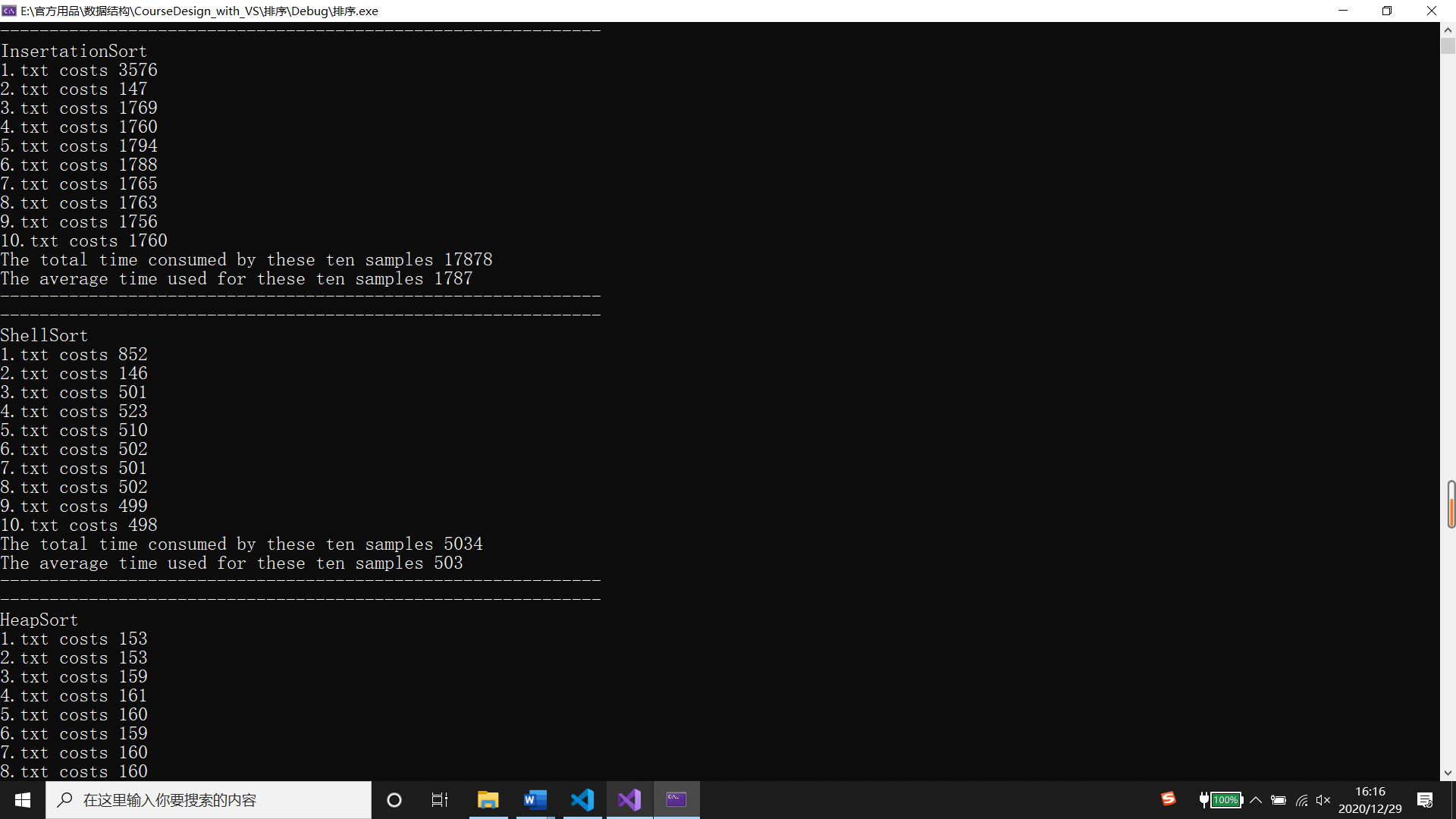
（1）实现交换、选择、归并等简单排序算法；

（2）实现快速排序算法；

（3）实现堆排序算法；

算法思想：了解熟悉各种基础排序算法

运行结果：



结果分析：运行结果正确，符合预期

附源程序。

#pragma comment(linker, "/STACK:10240000000,10240000000")

#include <iostream>

#include <ctime>

#include <string>

#include <sstream>

#include <cstdlib>

#include <fstream>

#include <algorithm>

using namespace std;

bool cmp(const int& a, const int& b)

{

return a > b;

}

void swap(int& a, int& b)

{

int temp = a;

a = b;

b = temp;

return;

}

void Generate\_random\_sample()

{

string fileName;

fstream file;

stringstream order;

int\* num1, \* num2;

num1 = (int\*)malloc(sizeof(int) \* 50010);

num2 = (int\*)malloc(sizeof(int) \* 50010);

for (int i = 3; i <= 10; i++)

{

order << i;

fileName = order.str() + ".txt";

cout << fileName << " " << i << " " << order.str() << endl;

file.open(fileName.c\_str(), ios::out);

if (!file)

{

cout << fileName << "can't be opened, please check it ！" << endl;

exit(0);

}

srand((unsigned int)time(NULL));

for (int j = 1; j <= 50000; j++)

{

file << rand() << " ";

if (j % 25 == 0)

{

file << endl;

}

}

order.str("");

file.close();

}

srand((unsigned int)time(NULL));

for (int i = 1; i <= 50000; i++)

{

num1[i] = rand();

num2[i] = num1[i];

}

sort(num1 + 1, num1 + 50000 + 1); //正序

sort(num2 + 1, num2 + 50000 + 1, cmp); //逆序

file.open("1.txt", ios::out);

for (int i = 1; i <= 50000; i++)

{

file << num2[i] << " ";

if (i % 25 == 0)

{

file << endl;

}

}

file.close();

file.open("2.txt", ios::out);

for (int i = 1; i <= 50000; i++)

{

file << num1[i] << " ";

if (i % 25 == 0)

{

file << endl;

}

}

file.close();

return;

}

void show(int a[], int n)

{

for (int i = 1; i <= n; i++)

{

cout << a[i] << " ";

}

cout << endl;

}

//a is the array, n is number, and first is 1

void Insertion\_sort(int a[], int n)

{

int j = 0, temp = 0;

for (int i = 2; i <= n; i++)

{

j = i;

temp = a[i];

while (j > 1 && a[j - 1] > temp)

{

a[j] = a[j - 1];

j--;

}

a[j] = temp;

}

}

//d is interval

void ShellInsertSort(int a[], int n, int d)

{

int temp, j;

for (int i = d + 1; i <= n; i++)

{

if (a[i] < a[i - d])

{

temp = a[i];

for (j = i; j > d; j = j - d)

{

if (temp < a[j - d])

{

a[j] = a[j - d];

}

else

{

break;

}

}

a[j] = temp;

}

}

}

void Shell\_sort(int a[], int n)

{

for (int i = 5; i >= 1; i -= 2)

{

ShellInsertSort(a, n, i);

//cout << i<<end

}

return;

}

void Bubble\_sort(int a[], int n)

{

for (int i = n; i > 1; i--)

{

for (int j = 1; j < i; j++)

{

if (a[j] > a[j + 1])

{

swap(a[j], a[j + 1]);

}

}

}

return;

}

int Partition(int a[], int low, int high)

{

int temp = a[low];

while (low < high)

{

while (low < high && a[high] >= temp)

{

high--;

}

a[low] = a[high];

while (low < high && a[low] <= temp)

{

low++;

}

a[high] = a[low];

}

a[low] = temp;

return low;

}

void Qsort(int a[], int low, int high)

{

int pivotloc;

if (low < high)

{

pivotloc = Partition(a, low, high);

Qsort(a, low, pivotloc - 1);

Qsort(a, pivotloc + 1, high);

}

return;

}

void Quick\_sort(int a[], int n)

{

Qsort(a, 1, n);

return;

}

void Selection\_sort(int a[], int n)

{

int k;

for (int i = 1; i <= n; i++)

{

k = i;

for (int j = i + 1; j <= n; j++)

{

if (a[k] > a[j])

{

k = j;

}

}

if (i != k)

{

swap(a[i], a[k]);

}

}

return;

}

void HeapAdjust(int a[], int low, int high)

{

int temp = a[low];

int i = low, j = i \* 2;

while (j < high)

{

if (j + 1 < high && a[j] < a[j + 1])

{

j++;

}

//在左右子树里边选择大的，j指向这个

if (temp >= a[j])

{

break;

}

//已经是大顶堆,结束这次循环

a[i] = a[j];

i = j;

j = 2 \* i;

}

a[i] = temp;

return;

}

//HeapAdjust

void Heap\_sort(int a[], int n)

{

for (int i = n / 2; i > 0; i--)

{

HeapAdjust(a, i, n); //从下往上调整一遍

}

for (int i = n; i > 1; i--)

{

swap(a[1], a[i]);

HeapAdjust(a, 1, i - 1);

}

return;

}

void MergeOneTime(int a[], int b[], int i, int m, int n)

{

int j, k;

for (j = m + 1, k = i; i <= m && j <= n; k++)

{

if (a[i] < a[j])

{

b[k] = a[i];

i++;

}

else

{

b[k] = a[j];

j++;

}

}

while (i <= m)

{

b[k++] = a[i++];

}

while (j <= n)

{

b[k++] = a[j++];

}

return;

}

//一趟归并，把a[i,m],a[m+1,n],归并为有序的b[i,n]

void Msort\_Re(int a[], int b[], int s, int t)

{

int\* c;

c = (int\*)malloc(sizeof(int) \* (t - s) + 10);

if (s == t)

{

b[s] = a[s];

}

else

{

int mid = (s + t) / 2;

Msort\_Re(a, c, s, mid);

Msort\_Re(a, c, mid + 1, t);

MergeOneTime(c, b, s, mid, t);

}

return;

}

//递归进行归并排序a[s,t],并入到b[s,t]

void MergeSort(int a[], int n)

{

Msort\_Re(a, a, 1, n);

return;

}

//归并排序

void MergePass(int a[], int b[], int k, int n)

//k为当前子序列的长度

{

int i, j;

for (i = 1; i <= n - 2 \* k + 1; i += k \* 2)

{

MergeOneTime(a, b, i, i + k - 1, i + 2 \* k - 1);

}

if (i + k < n)

{

MergeOneTime(a, b, i, i + k - 1, n);

}

else

{

for (j = i; j <= n; j++)

{

b[j] = a[j];

}

}

return;

}

//k是要一个子序列的长度，n是总长度

void Merge\_sort(int a[], int n)

{

int k = 1;

int\* b = (int\*)malloc(sizeof(int) \* n + 10);

while (k < n)

{

MergePass(a, b, k, n);

k \*= 2;

MergePass(b, a, k, n);

k \*= 2;

}

free(b);

return;

}

int CaculateMaxBit(int a[], int n)

{

int maxnum = a[1], maxbit = 0;

for (int i = 2; i <= n; i++)

{

if (maxnum < a[i])

{

maxnum = a[i];

}

}

while (maxnum)

{

maxbit++;

maxnum /= 10;

}

return maxbit;

}

//求出一组数据最大位数的函数

int GetSpecifiedBit(int num, int d)

{

int pow = 1;

d--;

while (d > 0)

{

pow \*= 10;

d--;

}

num /= pow;

num %= 10;

return num;

}

//取出数字num第d位的数字(注意是 从右向左数)

typedef struct node

{

int data;

struct node\* next;

}RadixNode;

void Radix\_sort(int a[], int n)

{

if (n <= 1)

{

return; //就一个或者零个数，还排个屁啊

}

RadixNode\* front[10], \* rear[10], \* head, \* tail = NULL, \* p = NULL; //十个队列的队头和队尾的指针

int max\_bit, temp;

max\_bit = CaculateMaxBit(a, n); //找出最大的位数,这决定了要进行几次的收集和发放

head = (RadixNode\*)malloc(sizeof(RadixNode));

head->data = -1;

head->next = NULL;

p = head;

for (int i = 1; i <= n; i++)

{

p->next = (RadixNode\*)malloc(sizeof(RadixNode));

p = p->next;

p->data = a[i];

p->next = NULL;

}//这样就建造好了一条链子

for (int j = 1; j <= max\_bit; j++)

{

for (int k = 0; k < 10; k++)

{

front[k] = NULL;

rear[k] = NULL;

}

//初始化

p = head->next;

while (p != NULL)

{

temp = GetSpecifiedBit(p->data, j);

if (front[temp] == NULL)

{

front[temp] = p;

rear[temp] = p;

}

else

{

rear[temp]->next = p;

rear[temp] = p;

}

p = p->next;

}

//断裂这条链子，然后放到桶中

head->next = NULL;

tail = NULL;

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

{

if (front[i] != NULL)

{

if (head->next == NULL)

{

head->next = front[i];

tail = rear[i];

continue;

}

else

{

tail->next = front[i];

tail = rear[i];

}

}

}

tail->next = NULL;

//从前到后进行收集

}

p = head->next;

for (int i = 1; i <= n && p != NULL; i++, p = p->next)

{

a[i] = p->data;

}

return;

}

bool FetchData(int a[], int i)

{

string fileName;

fstream file;

stringstream order;

order << i;

fileName = order.str() + ".txt";

//cout << fileName<<" " << i <<" "<<order.str()<<endl;

file.open(fileName.c\_str(), ios::in);

if (!file)

{

cout << fileName << "can't be opened, please check it ！" << endl;

exit(0);

}

for (int j = 1; j <= 50000; j++)

{

if (file.eof())

{

return false;

}

file >> a[j];

}

return true;

}

//取出第i个文件存放的数据,50000个都取出来

bool OneFunctionTime(void (\*p)(int\*, int))

{

int\* a;

int sum\_time = 0;

double av = 0;

a = (int\*)malloc(sizeof(int) \* 50010);

int n = 50000;

clock\_t start, end;

for (int i = 1; i <= 10; i++)

{

start = clock();

FetchData(a, i);

p(a, 50000);

end = clock();

//show(a, 10);

cout << i << ".txt costs " << end - start << endl;

sum\_time += (end - start);

}

av = sum\_time / 10;

cout << "The total time consumed by these ten samples " << sum\_time << endl;

cout << "The average time used for these ten samples " << av << endl;

return true;

}

void DisplayCaculationTime()

{

void (\*pfun)(int\*, int);

pfun = NULL; //pfun is a function pointer

cout << "-------------------------------------------------------------" << endl;

cout << "InsertationSort" << endl;

pfun = Insertion\_sort;

OneFunctionTime(pfun);

cout << "-------------------------------------------------------------" << endl;

cout << "-------------------------------------------------------------" << endl;

cout << "ShellSort" << endl;

pfun = Shell\_sort;

OneFunctionTime(pfun);

cout << "-------------------------------------------------------------" << endl;

cout << "-------------------------------------------------------------" << endl;

cout << "HeapSort" << endl;

pfun = Heap\_sort;

OneFunctionTime(pfun);

cout << "-------------------------------------------------------------" << endl;

cout << "-------------------------------------------------------------" << endl;

cout << "RadixSort" << endl;

pfun = Radix\_sort;

OneFunctionTime(pfun);

cout << "-------------------------------------------------------------" << endl;

cout << "-------------------------------------------------------------" << endl;

cout << "MergeSort" << endl;

pfun = Merge\_sort;

OneFunctionTime(pfun);

cout << "-------------------------------------------------------------" << endl;

cout << "-------------------------------------------------------------" << endl;

cout << "QuickSort" << endl;

pfun = Quick\_sort;

OneFunctionTime(pfun);

cout << "-------------------------------------------------------------" << endl;

cout << "-------------------------------------------------------------" << endl;

cout << "SelectionSort" << endl;

pfun = Selection\_sort;

OneFunctionTime(pfun);

cout << "-------------------------------------------------------------" << endl;

cout << "-------------------------------------------------------------" << endl;

cout << "BubbleSort" << endl;

pfun = Bubble\_sort;

OneFunctionTime(pfun);

cout << "-------------------------------------------------------------" << endl;

}

int main()

{

//Generate\_random\_sample();

DisplayCaculationTime();

return 0;

}

......

3、

题目：一并在第2题中

算法思想：

运行结果：

结果分析：

附源程序。

4、

题目：一并在第二题

算法思想：

运行结果：

结果分析：

附源程序。

5、

题目：一并在第二题

题目背景：

开学了，可是校园里堆积了不少垃圾杂物。

热心的同学们纷纷自发前来清理，为学校注入正能量～

题目描述：

通过无人机航拍我们已经知晓了n处尚待清理的垃圾位置，其中第i (1≤i≤n)处的坐标为(xi,yi)，保证所有的坐标均为整数。

我们希望在垃圾集中的地方建立些回收站。具体来说，对于一个位置(x,y)是否适合建立回收站，我们主要考虑以下几点：

1. (x,y)必须是整数坐标，且该处存在垃圾；
2. 上下左右四个邻居位置，即(x,y+1)、(x,y-1)、(x+1,y)和(x-1,y)处，必须全部存在垃圾；
3. 进一步地，我们会对满足上述两个条件的选址进行评分分数为不大于4的自然数，表示在(x±1,y±1)四个对角位置中有几处存在垃圾。

现在，请你统计一下每种得分的选址个数。

算法思想：按照要求，统计每个有效点周围的有效点个数，决定这个点是否可以选择为垃圾站，如果可以的话，就计算这个点的得分，最后统计答案。

运行结果：



结果分析：运行结果正确，通过CSP模拟测试。

附源程序。

#include <iostream>

#include <cstdlib>

using namespace std;

typedef struct node

{

int x;

int y;

int flag;

int value;

}Point;

int n;

Point spot[1005];

int ans[5];

bool check(Point &p)

{

for(int i = 1; i <= n ;i++)

{

if(spot[i].x == p.x - 1 && spot[i].y == p.y)

{

p.flag ++;

}

if(spot[i].x == p.x + 1 && spot[i].y == p.y)

{

p.flag ++;

}

if(spot[i].x == p.x && spot[i].y == p.y - 1)

{

p.flag ++;

}

if(spot[i].x == p.x && spot[i].y == p.y + 1)

{

p.flag ++;

}

if(spot[i].x == p.x - 1 && spot[i].y == p.y - 1)

{

p.value ++;

}if(spot[i].x == p.x + 1 && spot[i].y == p.y + 1)

{

p.value ++;

}if(spot[i].x == p.x - 1 && spot[i].y == p.y + 1)

{

p.value ++;

}if(spot[i].x == p.x + 1 && spot[i].y == p.y - 1)

{

p.value ++;

}

}

if(p.flag == 4)

{

return true;

}

else

{

return false;

}

}

int main()

{

//freopen("5.txt","r",stdin);

cin >> n;

for(int i = 1; i <= n; i++)

{

cin >> spot[i].x >> spot[i].y;

}

for(int i = 1; i <= n; i++)

{

if(check(spot[i]))

{

ans[spot[i].value] ++;

}

}

for(int i = 0; i < 5; i++)

{

cout << ans[i] <<endl;

}

return 0;

}

二、未调试成功程序及说明

1、

题目：

算法思想：

错误原因：

附源程序。

2、

题目：

算法思想：

错误原因：

附源程序。

......

三、代码行数及小结

