1.题目：实现二叉排序树的插入和删除

代码行数：221

源代码：

#include<iostream>

using namespace std;

class BSTNode{

public:

int key;

BSTNode \*lchild;

BSTNode \*rchild;

};

//插入函数 递归 该函数用T来表示整棵树的指针 返回值表示的是指向已经插入后的树的指针 因此就为T本身

BSTNode\* BSTInsert\_recursion(BSTNode \*T,int key){

//递归终止条件是 传入的根节点指向空

//也可以理解为 如果递归开始时 根节点为空 就直接插入

if(T==NULL){

//创建节点插入后返回

T=new BSTNode;

T->key=key;

//插入的一定是叶子节点

T->lchild=T->rchild=NULL;

return T;

}

else if(key==T->key){

return T;//如果在BST中找到了该值 无需插入 直接返回

}

else if(key<T->key){

//如果比当前节点值小 进入左子树递归 注意此处插入完后要将T->lchild赋值成该递归函数的返回值(表示更新成已经插入完成的左子树)

T->lchild=BSTInsert\_recursion(T->lchild, key);

return T;

}

else{//这里为啥只能以else结尾才不报错……

T->rchild=BSTInsert\_recursion(T->rchild, key);

return T;

}

}

//插入函数 非递归

BSTNode\* BSTInsert\_unRecursion(BSTNode \*T,int key){

//找到插入位 注意此处需要一个双亲指针来对插入位进行记录

BSTNode \*p,\*pa;

//初始化

p=T;

pa=NULL;//双亲指针一开始为空

while(p){//终止条件是p为空

//比较p所指的值和key

if(p->key==key){

//找到了相等值 不插入 直接返回

return T;

}

else if(key<p->key){

//向左子树偏移

//先偏移双亲指针

pa=p;

p=p->lchild;

}

else{

pa=p;

p=p->rchild;

}

}

//该循环结束后 p即指向插入位 pa即为p的双亲指针

//但此时p应该插左还是插右尚没有确定 所以应当根据pa所指的值来进行判断

//先创建节点

p=new BSTNode;

p->key=key;

//插入的一定是叶子节点

p->lchild=p->rchild=NULL;

//如果双亲不存在 表明一开始的根节点为空 创建完p后 此时相当于已经插入 直接返回即可

if(pa==NULL){

return T;

}

//双亲存在时 要和pa所指的值进行比较 确定放在哪一个子树 p放置完毕后记得要返回T

else if(p->key<pa->key){

pa->lchild=p;

return T;

}

else{

pa->rchild=p;

return T;

}

}

//删除函数 返回值为指向删除后树的指针T

BSTNode\* BSTDelete(BSTNode \*T,int key){

//查找删除位置 p记录删除位 pa记录删除位双亲

BSTNode \*p=T;

BSTNode \*pa=NULL;

while(p){

//如果找到 退出循环

if(p->key==key){

break;

}

else if(key<p->key){

//左分支

pa=p;

p=p->lchild;

}

else{

//右分支

pa=p;

p=p->rchild;

}

}

//退出循环时 如果p为空 表示未找到 直接返回原树

if(p==NULL){

return T;

}

//能够执行到下面 说明已经找到删除位 开始删除

//1.叶子节点

if(p->lchild==NULL&&p->rchild==NULL){

//注意 根节点由于双亲为空 操作与其他点有不同 需要单独考虑

if(pa==NULL){

//双亲为空 表示p为根节点

//直接删除p 后由于根节点被删除且根结点为叶子节点 所以此时树为空 返回NULL即可(返回此时的T也可以）

delete p;

return NULL;

}

else{

//p不为根节点

//先判断p是双亲节点的哪边子树上的点 然后对应删除即可

if(p->key<pa->key){

//左子树

pa->lchild=NULL;

delete p;

return T;

}

else{

//右子树 此处无需使用else if语句是因为不可能出现相等的情况

pa->rchild=NULL;

delete p;

return T;

}

}

}

//2.单枝节点 将单枝上移即可 注意也要考虑根节点

//具有左单枝

else if(p->lchild&&!p->rchild){

if(pa==NULL){

//p为根节点 直接将根节点覆盖为自己的左子树(左子树的根节点成为新的根) 然后删除p

//注意此处要对T而不是p赋值 p用于删除

T=p->lchild;

delete p;

return T;

}

else{

//若不为根节点 另pa原来指向p的位置指向p的左枝

if(p==pa->lchild){

pa->lchild=p->lchild;

delete p;

return T;

}

else{

pa->rchild=p->lchild;

delete p;

return T;

}

}

}

//具有右单枝

else if(!p->lchild&&p->rchild){

if(pa==NULL){

//p为根节点 直接将根节点覆盖为自己的左子树(左子树的根节点成为新的根) 然后删除p

//注意此处要对T而不是p赋值 p用于删除

T=p->rchild;

delete p;

return T;

}

else{

//若不为根节点 另pa原来指向p的位置指向p的左枝

if(p==pa->lchild){

pa->lchild=p->rchild;

delete p;

return T;

}

else{

pa->rchild=p->rchild;

delete p;

return T;

}

}

}

//3.双枝节点

//先找到左子树的最右节点 然后另该点覆盖需删除点的key 最后删除最右节点

//用q表示最右节点 qa表示q的双亲节点 注意q只有可能是叶子节点或只有左枝

else{

BSTNode \*qa=p;

BSTNode \*q=p->lchild;

while(q->rchild){

//只要q还存在右子树 就一直遍历

qa=q;

q=q->rchild;

}

//循环结束后 q为最右节点

//先令q覆盖p

p->key=q->key;

//然后分情况删除q 注意q不为根 无需考虑特殊情况

if(!q->lchild&&!q->rchild){

//叶子节点

if(q==qa->lchild){

qa->lchild=NULL;

delete q;

}

else{

qa->rchild=NULL;

delete q;

}

return T;

}

else{

//只有可能是q存在左子树

if(q==qa->lchild){

qa->lchild=q->lchild;

delete q;

}

else{

qa->rchild=q->lchild;

delete q;

}

return T;

}

}

}

int main(){

return 0;

}

2.题目：实现交换、选择、归并等简单排序算法；

代码行数：113

源代码：

#include<iostream>

#include<vector>

using namespace std;

//均默认为从小到大排序

//已测试

void bubble\_sort(){

vector<int> data{48,35,66,91,74,18,22,48,57,3};//存储未排序的初始数据

for(int i=0;i<data.size()-1;i++){

//趟数 只需进行到n-1趟

for(int j=0;j<data.size()-1-i;j++){

//大数下沉

if(data[j]>data[j+1]){

int temp=data[j];

data[j]=data[j+1];

data[j+1]=temp;

}

}

}

for(int i=0;i<data.size();i++){

cout<<data[i]<<" ";

}

cout<<endl;

}

//选择排序

void select\_sort(){

vector<int> data{48,35,66,91,74,18,22,48};

for(int i=0;i<data.size()-1;i++){

int min=data[i];

int indexOfMin=i;//将最小值初始化为第一项

//比较的起始数只需进行到倒数第二项

for(int j=i+1;j<data.size();j++){

//从当前起始数的后一位开始 比较到末尾

if(data[j]<min){

min=data[j];

indexOfMin=j;

}

}

//此时已找到最小值 之后进行交换即可

int temp=data[i];

data[i]=data[indexOfMin];

data[indexOfMin]=temp;

}

for(int i=0;i<data.size();i++){

cout<<data[i]<<" ";

}

cout<<endl;

}

//归并两路函数

void merge(vector<int> &data,int low,int mid,int high){

//归并线性表data中 [low,mid]和[mid+1,high]的数据

vector<int> tempSq;

int i=low;

int j=mid+1;

while(i<=mid&&j<=high){

if(data[i]<=data[j]){

tempSq.emplace\_back(data[i]);

i++;

}

else{

tempSq.emplace\_back(data[j]);

j++;

}

}

//跳出该函数后 有可能还有一些剩余的元素 直接放到后面即可

while(i<=mid){

tempSq.emplace\_back(data[i]);

i++;

}

while(j<=high){

tempSq.emplace\_back(data[j]);

j++;

}

//此时tempSq中存放着指定区间已排序的数据 将这些数据重新赋回到原data中

i=low;

for(int k=0;k<tempSq.size();k++){

data[i]=tempSq[k];

i++;

}

}

//一趟归并将两两长度为len的子序列归并

void mergeLen(vector<int> &data,int len){

int i=0;//表示相邻长度为len的两个子序列的首元素处

while(i+2\*len<data.size()){

//当后续还有两个完整的长度为len的序列时 直接用len对这两个区间进行分割

merge(data,i,i+len-1,i+2\*len-1);

//当前归并后记得偏移 i偏移至下两个完整len序列的开头

i=i+2\*len;

}

//退出循环时 有可能出现剩下的数中 有一个完整的len序列 还有一个不足长度不足len的序列

//直接将最后一个len序列和剩下的所有元素作归并即可

if(i+len<data.size()){

merge(data,i,i+len-1,data.size()-1);

}

}

//归并排序函数

void mergeSort(){

vector<int> data{48,35,66,91,74,18,22,48,57,3};

for(int len=1;len<=data.size();len\*=2){

//最开始长度为1 长度每次变为原来两倍 直至len已经达到最逼近数据总数的情况

mergeLen(data, len);

}

//测试

for(int i=0;i<data.size();i++){

cout<<data[i]<<" ";

}

cout<<endl;

}

int main(){

// bubble\_sort();

// select\_sort();

mergeSort();

return 0;

}

3.题目：实现快速排序算法

代码行数：47

源代码：

#include<iostream>

#include<vector>

using namespace std;

//分割函数 前面的部分全小于某值 后面的部分全大于某值 返回值为最终枢纽的位置

int partition(vector<int> &data,int low,int high){

int center=data[low];

while(low<high){

while(low<high&&data[high]>=center){

//注意一定要加low<high的条件 因为high和low可能会在大的while语句内部就出现交叉的情况

high--;

}

//如果跳出上面循环 表示找到了不符合要求的值 将其移到前面

data[low]=data[high];

//此处的移动用赋值代替 并没有实际意义上的先删除再移动

while(low<high&&data[low]<=center){

low++;

}

data[high]=data[low];

}

//退出上面的循环时 low==high即两者均表示中间位 此时要将枢纽值赋值给中间位

data[low]=center;

return low;

}

//对指定区间快速排序函数

void qSort(vector<int> &data,int low,int high){

//注意要加限制以使递归终止

if(low<high){

int center=partition(data,low,high);//先对当前序列分割

//再分别对前面和后面的序列进行递归快速排序

qSort(data,low,center);

qSort(data,center+1,high);

}

}

//快速排序函数

void quickSort(){

vector<int> data{48,35,66,91,74,18,22,48,57,3};

qSort(data, 0, data.size()-1);

//测试

for(int i=0;i<data.size();i++){

cout<<data[i]<<" ";

}

cout<<endl;

}

int main(){

quickSort();

return 0;

}

4.题目：实现堆排序算法

代码行数：57

源代码：

#include<iostream>

#include<vector>

using namespace std;

//堆调整算法

//注意该操作是在对全堆进行调整中的一次基本操作

//即假设左右子树全部为大顶堆的前提下进行的操作

void heapAdjust(vector<int> &data,int low,int high){

int codeOfKeyTocheck=low;//创建变量存储当前比较值的编号

int codeOfBiggerChild=2\*codeOfKeyTocheck;//创建变量存储较大子节点的编号 初始化为左子节点

while(codeOfBiggerChild<=high){

//左子节点仍然存在时 尚未进行到叶子 继续比较

//首先与可能存在的右子节点进行比较取较大值

if(codeOfBiggerChild+1<=high&&data[codeOfBiggerChild+1]>data[codeOfBiggerChild]){

//右子树存在且右子节点比左子节点更大

codeOfBiggerChild++;

}

//首先进行一次与最大子节点的比较 一旦更大即可直接退出 已为大顶堆

if(data[codeOfKeyTocheck]>data[codeOfBiggerChild]){

break;

}

//若能进行到下面 表明子节点中存在比自己大的值 要进行与子节点的交换

int temp=data[codeOfKeyTocheck];

data[codeOfKeyTocheck]=data[codeOfBiggerChild];

data[codeOfBiggerChild]=temp;

//完成交换后 需要向下进行偏移

codeOfKeyTocheck=codeOfBiggerChild;

codeOfBiggerChild=2\*codeOfKeyTocheck;

}

}

void heapSort(vector<int> &data){

//首先从最后一个非终端节点n//2开始 到1结束 将整个数组调整为一个大顶堆

int n=data.size()-1;

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

//注意此处的n/2就是整除

heapAdjust(data, i, n);

}

//调整为大顶堆后 依次取最末尾的值与顶端值交换 然后对剩下的值再进行一次堆调整

//注意该操作只需进行到倒数第二个元素 剩下来的两个元素一旦交换即排序完成

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

int temp=data[1];

data[1]=data[i];

data[i]=temp;

heapAdjust(data, 1, i-1);

}

//测试

for(int i=1;i<data.size();i++){

cout<<data[i]<<" ";

}

cout<<endl;

}

int main(){

//测试

//注意二叉树编号是从1开始的 否则将不满足一般的编号规律 0号位置的空间要舍弃

vector<int> data{-1,48,35,66,91,74,18,22,48};

heapSort(data);

return 0;

}

5.题目：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：

a.  (x,y)必须是整数坐标，且该处存在垃圾；

b.  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；

c.  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：无特殊算法 首先统计出可以做垃圾站的点 然后统计出分数

代码行数：70

源代码：#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

//统计垃圾点数函数 注意统计分数是要在能做垃圾站的点里进行的 且统计的点是对角线

int countTrash(int x,int y,vector<pair<int,int>> &trashPlace){

int count=0;

if(find(trashPlace.begin(),trashPlace.end(),pair<int,int>{x+1,y+1})!=trashPlace.end()){

count++;

}

if(find(trashPlace.begin(),trashPlace.end(),pair<int,int>{x-1,y-1})!=trashPlace.end()){

count++;

}

if(find(trashPlace.begin(),trashPlace.end(),pair<int,int>{x+1,y-1})!=trashPlace.end()){

count++;

}

if(find(trashPlace.begin(),trashPlace.end(),pair<int,int>{x-1,y+1})!=trashPlace.end()){

count++;

}

return count;

}

bool isOK(int x,int y,vector<pair<int,int>> &trashPlace){

//以下四个点只要有一个没有找到 就返回false

if(find(trashPlace.begin(),trashPlace.end(),pair<int,int>{x,y+1})==trashPlace.end()){

return false;

}

if(find(trashPlace.begin(),trashPlace.end(),pair<int,int>{x,y-1})==trashPlace.end()){

return false;

}

if(find(trashPlace.begin(),trashPlace.end(),pair<int,int>{x+1,y})==trashPlace.end()){

return false;

}

if(find(trashPlace.begin(),trashPlace.end(),pair<int,int>{x-1,y})==trashPlace.end()){

return false;

}

//如果前面都没有返回 则表示四个点都有 返回true

return true;

}

void solution(vector<int> &score,vector<pair<int,int>> &trashPlace){

vector<pair<int,int>> OKPlaces;

for(int i=0;i<trashPlace.size();i++){

//首先统计出可以做垃圾站的点

bool flag=isOK(trashPlace[i].first, trashPlace[i].second, trashPlace);

if(flag==true){

OKPlaces.push\_back(trashPlace[i]);

}

}

//然后统计分数

for(int i=0;i<OKPlaces.size();i++){

int count=countTrash(OKPlaces[i].first, OKPlaces[i].second, trashPlace);

score[count]++;

}

}

int main(){

int n;

cin>>n;

vector<pair<int,int>> trashPlace;

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

int x,y;

cin>>x>>y;

trashPlace.emplace\_back(x,y);

}

vector<int> score;

score.resize(5);//全部初始化为0

solution(score, trashPlace);

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

cout<<score[i]<<endl;

}

return 0;

}