**《数据结构》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2020级计算机科学与技术01班** | | | **姓名** | **陈鹏宇** |
| **实验题目** | **二叉树的应用** | | | | | |
| **实验时间** | **2021/11/23** | | **实验地点** | **Ds1408** | | |
| **实验成绩** |  | | **实验性质** | **□验证性** √**设计性 □综合性** | | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  1. 学习并掌握二叉树数据结构及实现方法，掌握树的遍历方法。  2. 熟练掌握二叉检索树（BST）的基本操作及其应用。 | | | | | | |
| 二、实验项目内容   1. 利用BST实现一个城市数据库：每个数据库结点包括城市名称和以整数x与y表示的城市坐标，根据城市名称组织该BST； 2. 在该数据库上实现按城市名称进行的插入、删除和检索； 3. 打印出以指定字母打头的所有城市记录； 4. 打印出与指定点的距离在给定值之内的所有城市记录； 5. 最后提交完整的实验报告和源程序。 | | | | | | |
| 1. 实验过程或算法（源程序）   #include <bits/stdc++.h>  using namespace std;  struct city{ //构造新的city结构，包含城市名和对应横纵坐标  string name;  int x;  int y;  };  class node{ //bst树节点型  private:  city it;  node\* lc;  node\* rc;  public:  node() {lc = rc = nullptr; }  node(city c , node\* l , node\* r){  it = c ; lc = l ; rc = r;  }  ~node(){}  string getName(){ return it.name; } //返回城市名  int getX(){ return it.x; } //返回城市横坐标  int getY(){ return it.y; } //返回城市纵坐标  city getCity(){ return it; }  void setCity(city temp){ //初始化城市  it.name = temp.name;  it.x = temp.x;  it.y = temp.y;  }  inline node\* left() const{return lc;}  inline node\* right() const{return rc;}  void setLeft(node\* b){lc = b;}  void setRight(node\* b){rc = b;}  bool isleaf(){return (lc == nullptr) && (rc == nullptr);}  };  class bst{  private:  node\* root;  int nodecount;  void clearhelp(node\* root){  if(root == nullptr) return;  clearhelp(root->left());  clearhelp(root->right());  delete root;  }  node\* inserthelp(node\* root , city temp){  if(root == nullptr)  return new node (temp , nullptr , nullptr);  if(temp.name < root->getName()){  root->setLeft(inserthelp(root->left() , temp));  }  else{  root->setRight(inserthelp(root->right() , temp));  }  return root;  }  node\* deletemin(node\* rt){  if(rt->left() == nullptr)  return rt->right();  else{  rt->setLeft(deletemin(rt->left()));  return rt;  }  }  node\* getmin(node\* rt){  if(rt->left() == nullptr)  return rt;  return getmin(rt->left());  }  node\* removehelp(node\* rt , string s){  if(rt == nullptr) return nullptr;  else if(s < rt->getName())  rt->setLeft(removehelp(rt->left() , s));  else if(s > rt->getName())  rt->setRight(removehelp(rt->right() , s));  else{  node\* temp = rt;  if(rt->left() == nullptr){  rt = rt->right();  delete temp;  }  else if(rt->right() == nullptr){  rt = rt->left();  delete temp;  }  else{  node\* temp = getmin(rt->right());  rt->setCity(temp->getCity());  rt->setRight(deletemin(rt->right()));  delete temp;  }  }  return rt;  }  string findhelp(node\* root , string s){  if(root == nullptr) return NULL;  if(s < root->getName())  return findhelp(root->left() , s);  else if(s > root->getName())  return findhelp(root->right() , s);  else return root->getName();  }  void printhelp(node\* root){  if(root == nullptr) return;  printhelp(root->left());  cout<<root->getName()<<endl;  printhelp(root->right());  }  public:  bst() { root = nullptr ; nodecount = 0; }  ~bst() { clearhelp(root); }  node\* getRoot() {return root; }  void clear() { clearhelp(root); root = nullptr; nodecount = 0; }  void insert(city c)  {  root = inserthelp(root, c);  nodecount++;  }  string remove(string s)  {  string temp = findhelp(root, s);  if (temp.c\_str() != NULL)  {  root = removehelp(root, s);  nodecount--;  }  return temp;  }  string find(string& k) { return findhelp(root, k); }  int size() { return nodecount; }  void print() //中序遍历城市，打印城市名  {  //打印BST的值  if (root == nullptr)  cout << "The BST is empty."<<endl;  else printhelp(root);  }  };  void mySearchByChar(node\* rt , char c){ //中序遍历搜索以c开头的城市  if(rt == nullptr) return;  mySearchByChar(rt->left() , c);  if(rt->getName()[0] == c)  cout<<rt->getName()<<" "<<rt->getX()<<" "<<rt->getY()<<endl;  mySearchByChar(rt->right() , c);  }  void mySearchByDis(node\* rt , int X , int Y , int dis){ //中序遍历搜索距离XY点小于dis的城市  if(rt == nullptr) return;  mySearchByDis(rt->left() , X , Y , dis);  if((pow(rt->getX()-X , 2) + pow(rt->getY()-Y , 2)) < dis\*dis){  cout<<rt->getName()<<" "<<rt->getX()<<" "<<rt->getY()<<endl;  }  mySearchByDis(rt->right() , X , Y , dis);  }  int main()  {  int citynum ; cin >> citynum; //初始城市数  bst\* tree = new bst();  for(int i = 0 ; i < citynum ; i++){ //初始化城市数据库  city temp;  cin >> temp.name >> temp.x >> temp.y;  tree->insert(temp);  }  int temp ; cin >> temp; //循环插入删除  while(temp != 2){  if(temp == 1){  city t ; cin >> t.name >> t.x >> t.y;  tree->insert(t);  }  else if(temp == 0){  string n ; cin >> n;  tree->remove(n);  }  cin >> temp;  }  tree->print(); //中序遍历  char c ; cin >> c;  mySearchByChar(tree->getRoot() , c); //搜索c开头的城市  int X , Y , dis ; cin >> X >> Y >> dis;  mySearchByDis(tree->getRoot() , X , Y , dis); //搜索距离XY小于dis的城市  return 0;  } | | | | | | |
| 1. 实验结果及分析和（或）源程序调试过程   实验结果    调试过程  由于书上程序太过复杂，我直接将KVpair 设置为一struct，代替了书上的模板类    其中以city的名字进行排序，且node类新增对city信息的修改和返回操作。  对书上代码进行相应改变后，通过中序遍历检验树是否生成正确  NW[D94XQ@WGR(V%1%LA2UX4  对于插入和删除，只需套用类中的成员函数即可。而检索功能，根据题意，就是以中序遍历为基础，打印符合规范的城市信息。 | | | | | | |