山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201700130009 | 姓名： 张愈博 | | 班级： 计科17.3 |
| 实验题目：搜索树 | | | |
| 实验学时：2 | | 实验日期： 2018.12.03 | |
| 实验目的：  掌握二叉搜索树结构的定义、描述方法、操作实现。 | | | |
| 软件环境：  Windows 10  Dev C++ 5.11 | | | |
| 1. 实验内容（题目内容，输入要求，输出要求）   （1）创建带索引的二叉搜索树类。存储结构使用链表，提供操作:插入、删除、  按名次删除、查找、按名次查找、升序输出所有元素。  （2）接收键盘录入的一系列整数，输出其对应的二叉搜索树（可使用文本形式  输出），以及二叉搜索树的高度。  （3）对建立好的二叉搜索树，执行上述各操作（插入操作除外），输出各操作  的结果（插入、删除操作输出完成后的二叉搜索树；查找操作输出查找过  程中依次比较的元素）。   1. 数据结构与算法描述（整体思路描述，所需要的数据结构与算法）   带索引的二叉树类继承自binaryTree，节点元素是Pair，内含3个变量：key, value和leftSize。以下是各方法的算法：  (1)插入操作。设一个指针p指向当前节点（初始化为root），pp指向p的父亲节点，然后根据要插入的节点的key与当前节点p相比较，不断更新p寻找插入位置（若为索引二叉树还要更新leftSize值），该位置不为空则覆盖，为空则创建一个新节点，与pp相连接。注意这里插入的一定是叶节点。  (2)删除操作。先用搜索算法找到与关键字相匹配的节点。若该节点是叶节点，直接删除并且将其父节点的指针置为空；若该节点有一棵子树，将其子树与父节点相连接然后删除该节点；若该节点有两颗子树，则从左子树中取一个最大元素放到该节点的位置上（并不是简单的移动），再把该最大元素删除（方法同上）。注意本实验是带索引二叉树的按关键字删除，所以为了之后的操作也要加更新根到节点的leftSize值。  (3)按名次删除。首先按名次搜索算法找出待删除元素的位置，然后先更新根到该节点的leftSize域，之后同（2）的删除操作。  (4)查找操作。给出关键字key，令p指向当前节点，若key小于p的key值则继续搜索左子树，大于则搜索右子树，等于则返回对应节点的指针，找不到则返回NULL。  (5)按名次查找。给出名次index，令p指向当前节点，若index小于p的leftSize值则继续搜索左子树，若index大于p的leftSize值则继续搜索右子树，并更新index为index-p.leftSize-1。若index等于p的leftSize则返回p。  (6)升序输出。将索引二叉树中序遍历输出即可。   1. 测试结果（测试输入，测试输出，结果分析）     图1 图2  图1是索引二叉树的建立过程、索引搜索、关键字查找和按索引删除。  图2是插入和按关键字删除操作。   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）     在调试过程中执行了删除关键字为4的元素操作，出现了如图的问题尚未解决。从程序运行结果看，delete操作根本没有成功。  在此步出现了断点，但是看不出问题所在。  经尝试，发现把这部分代码用注释内的代码替换就可以正常使用。经反复尝试发现我在一开始就把pp赋给了p，这样即使找到了相应数对，pp和p也是指向同一个地方的，这就不符合当初的设计思想。所以修改也很简单，只有在大于或者小于的时候才把pp赋给p。   1. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）   Main.cpp  #include<iostream>  #include"indexedBinarySearchTree.h"  #include"binarySearchTree.h"  using namespace std;  int main()  {  /\* binarySearchTree<int,int> bsT; \*/  indexedBinarySearchTree<int,int> ibsT;  cout<<"please input the number of pairs:"; int n; cin>>n;  int \*first=new int [n];  int \*second=new int [n];  Pair<int,int> \*P=new Pair<int,int> [n];  cout<<"please input the key and value of pairs:"<<endl;  for(int i=0;i<n;i++)  {  cin>>first[i];  cin>>second[i];  P[i].first=first[i];  P[i].second=second[i];    }  for(int i=0;i<n;i++)  {  ibsT.insert(P[i]);  }  ibsT.ascend();  cout<<"The height is:"<<ibsT.height()<<endl;  cout<<"Please input the index you want to find:"; cin>>n;  if(ibsT.find(n)!=NULL) cout<<"find succeed!"<<endl;  else cout<<"find failed!"<<endl;    cout<<"Please input the pair's key you want to find:"; cin>>n;  if(ibsT.find(n,true)!=NULL) cout<<"find succeed!"<<endl;  else cout<<"find failed!"<<endl;    cout<<endl<<endl;  cout<<"please input the index you want to erase:"; cin>>n;  ibsT.erase(n);  ibsT.ascend();  cout<<"The height is:"<<ibsT.height()<<endl;    cout<<"please input the data you want to insert:";  int key,value; cin>>key>>value;  Pair<int,int> temp(key,value);  ibsT.insert(temp);  ibsT.ascend(); cout<<endl;    cout<<"please input the pair's key you want to erase:"; cin>>n;  ibsT.erase(n,true);  ibsT.ascend();  return 0;  }  indexedBinarySearchTree.h  #pragma once  #include<iostream>  #include"二叉树.h"  #include"pair.h"  using namespace std;  template<class K,class E>  class indexedBinarySearchTree:public linkedBinaryTree<Pair<K, E> >  {  public:  typedef linkedBinaryTree< Pair<K, E> > BT;  indexedBinarySearchTree()  {    }  Pair<K,E>\* find(int index) const  {//²éÕÒµ½µÄÔªËØÊÇ×ÖµäÅÅÐòÖÐµÄµÚindex+1´óÔªËØ  binaryTreeNode< Pair<K,E> > \*p=BT::root;  while(p!=NULL)  {  cout<<"compare "<<index<<" with pair< "<<p->element<<" >'s leftSize:"<<p->element.leftSize<<endl;  if(index<p->element.leftSize)  p=p->left; //¼ÌÐøËÑË÷×ó×ÓÊ÷  else  if(index>p->element.leftSize)  {  index=index - (p->element.leftSize+1); //¸üÐÂindex  p=p->right;  }  else  return &(p->element); //ÕÒµ½ÁËÆ¥ÅäµÄÔªËØ  }  return NULL;  }  Pair<K,E>\* find(K &theKey,bool flag) const  {//·µ»ØÆ¥ÅäÊý¶ÔµÄÖ¸Õë£¬ÈôÎÞÆ¥ÅäÔò·µ»Ønull  binaryTreeNode< Pair<K,E> > \*p=BT::root;  while(p!=NULL)  {  cout<<"compare "<<theKey<<" with pair< "<<p->element<<">"<<endl;  if(theKey<p->element.first)  p=p->left; //¼ÌÐøËÑË÷×ó×ÓÊ÷  else  if(theKey>p->element.first)  p=p->right;  else  return &(p->element); //ÕÒµ½ÁËÆ¥ÅäµÄÔªËØ  }  return NULL;  }  void insert(Pair<K,E> &thePair)  {//Òª¸üÐÂ¸ùÖÁ²åÈë½ÚµãÂ·¾¶ÉÏµÄleftSizeÖµ  binaryTreeNode< Pair<K,E> > \*p=BT::root,  \*pp=NULL; //pp is the parent of p and trails p  while(p!=NULL)  {  pp=p;  if(thePair.first<p->element.first)  { p->element.leftSize++; p=p->left; } //²åÈëËüµÄ×ó×ÓÊ÷Ê±Òª¸üÐÂleftSize  else  if(thePair.first>p->element.first)  p=p->right;  else //ÕÒµ½¹Ø¼ü×ÖÏàÍ¬µÄÊý¶Ô£¬¸²¸Ç  {  p->element.second=thePair.second;  return;  }  }  //Ã»ÓÐÕÒµ½Êý¶Ô£¬ÄÇÃ´ÔÚ²åÈëÎ»ÖÃ½¨Á¢Ò»¸öÐÂ½Úµã£¬È»ºóÓëppÁ¬½Ó  binaryTreeNode< Pair<K,E> > \*newNode=new binaryTreeNode< Pair<K,E> >(thePair);  if(BT::root!=NULL) //Ê÷²»Îª¿Õ  {  if(thePair.first < pp->element.first) //thepairµÄ¹Ø¼ü×Ö±ÈppÐ¡£¬·Å×ó×ÓÊ÷  pp->left=newNode;  else  pp->right=newNode;  }  else  BT::root=newNode;  BT::treeSize++;  }  void erase(int index)  {//°´Ë÷ÒýÉ¾³ý  binaryTreeNode< Pair<K,E> > \*p=BT::root,  \*pp=NULL;  while(p!=NULL && p->element.leftSize != index)  {  pp=p;  if(index<p->element.leftSize)  p=p->left; //¼ÌÐøËÑË÷×ó×ÓÊ÷  else  {  index=index - (p->element.leftSize+1); //¸üÐÂindex  p=p->right;  }  }  if(p==NULL)  {  cerr<<"Î´ÕÒµ½Æ¥ÅäÊý¶Ô£¡"<<endl;  return; //Ã»ÓÐÆ¥ÅäÊý¶Ô  }  else  {//ËµÃ÷ÕÒµ½ÁË¶ÔÓ¦Êý¶Ô£¬ÏÈ¸üÐÂÒ»±é¸ùµ½pµÄleftSizeÖµ  binaryTreeNode< Pair<K,E> > \*t=BT::root;  while(t!=NULL)  {  if(index<t->element.leftSize)  {//¼ÌÐøËÑË÷×ó×ÓÊ÷  t->element.leftSize--;  t=t->left;  }  else  if(index>t->element.leftSize)  t=t->right;  else  break;  }  }    //ÏÂÃæÐèÒªÖØ¹¹Ê÷µÄ½á¹¹£¬Óë¶þ²æËÑË÷Ê÷²Ù×÷ÍêÈ«ÏàÍ¬  //µ±pÓÐÁ½¸öº¢×ÓÊ±  if(p->left!=NULL&&p->right!=NULL)  {//ÕÒµ½×ó×ÓÊ÷ÖÐ×î´óÔªËØ  binaryTreeNode< Pair<K,E> > \*s=p->left,  \*ps=p;  while(s->right!=NULL)  {//Ìø³öÑ­»·ºós¾ÍÖ¸ÏòÁË×ó×ÓÊ÷ÖÐ×î´óÔªËØ  ps=s;  s=s->right;  }    //½«×î´óÔªËØsÒÆµ½p£¬µ«²»ÊÇ¼òµ¥µÄÒÆ¶¯£¬»­¸öÊ¾ÒâÍ¼¾ÍÍ¦ºÃÀí½âµÄÁË  binaryTreeNode< Pair<K,E> > \*q=new binaryTreeNode< Pair<K,E> >(s->element,p->left,p->right);  if(pp==NULL)  BT::root=q;  else{//Á¬½ÓppÓëq  if(p==pp->left)  pp->left=q;  else pp->right=q;  }  if(ps==p) pp=q; //ÐèÒªµ¥¶ÀÌÖÂÛpµÄ×ó×ÓÊ÷ÊÇ·ñÖ»ÓÐÒ»¸öÔªËØ£¬ÒòÎªÈç¹ûps=old p£¬»á±»É¾³ýµÄ¡£  else pp=ps;  delete p; cout<<"p="<<p<<endl; //Ã²ËÆÉ¾³ý¸ù±¾¾Í²»³É¹¦£¿  p=s;    }//ÕâÊ±pÒÑ¾­Ö¸ÏòÒªÉ¾³ýµÄ×ó×ÓÊ÷×î´óÔªËØ    //µ±pÖ»ÓÐÒ»¸öº¢×Ó»òÕßÊÇÒ¶½ÚµãÊ±  binaryTreeNode< Pair<K,E> > \*c;  if(p->left!=NULL) //ÈÃcÖ¸Ïòp·Ç¿ÕµÄº¢×Ó  c=p->left;  else  c=p->right;    if(p==BT::root) c=BT::root;  else  {//ÕÒ³öpÊÇppµÄÄÄ¸öº¢×Ó  if(p==pp->left)  pp->left=c;  if(p==pp->right)  pp->right=c;  }  delete p; cout<<"p="<<p<<endl;  BT::treeSize--;  }  void erase(K theKey,bool flag)  {//É¾³ý¹Ø¼ü×ÖÊÇtheKeyµÄÊý¶Ô  binaryTreeNode< Pair<K,E> > \*p=BT::root,  \*pp=NULL;  while (p != NULL && p->element.first != theKey)  {// move to a child of p  pp = p;  if (theKey < p->element.first)  p = p->left;  else  p = p->right;  }  if(p==NULL) return; //Ã»ÓÐÆ¥ÅäÊý¶Ô  else  {//ËµÃ÷ÕÒµ½ÁË¶ÔÓ¦Êý¶Ô£¬ÏÈ¸üÐÂÒ»±é¸ùµ½pµÄleftSizeÖµ  cout<<endl<<"the p's element is:'"<<p->element<<endl<<endl;  binaryTreeNode< Pair<K,E> > \*t=BT::root;  while(t!=NULL)  {  if(theKey<t->element.first)  {//¼ÌÐøËÑË÷×ó×ÓÊ÷  t->element.leftSize--;  t=t->left;  }  else  if(theKey>t->element.first)  t=t->right;  else  break;  }  }    //ÏÂÃæÐèÒªÖØ¹¹Ê÷µÄ½á¹¹  //µ±pÓÐÁ½¸öº¢×ÓÊ±  if(p->left!=NULL&&p->right!=NULL)  {//ÕÒµ½×ó×ÓÊ÷ÖÐ×î´óÔªËØ  binaryTreeNode< Pair<K,E> > \*s=p->left,  \*ps=p;  while(s->right!=NULL)  {//Ìø³öÑ­»·ºós¾ÍÖ¸ÏòÁË×ó×ÓÊ÷ÖÐ×î´óÔªËØ  ps=s;  s=s->right;  }    binaryTreeNode< Pair<K,E> > \*q=new binaryTreeNode< Pair<K,E> >(s->element,p->left,p->right);  if(pp==NULL)  BT::root=q;  else{//Á¬½ÓppÓëq  if(p==pp->left)  pp->left=q;  else pp->right=q;  }  if(ps==p) pp=q; //ÐèÒªµ¥¶ÀÌÖÂÛpµÄ×ó×ÓÊ÷ÊÇ·ñÖ»ÓÐÒ»¸öÔªËØ£¬ÒòÎªÈç¹ûps=old p£¬»á±»É¾³ýµÄ¡£  else pp=ps;  delete p; p=NULL; cout<<"p="<<p<<endl; //Ã²ËÆÉ¾³ý¸ù±¾¾Í²»³É¹¦£¿  p=s;    }    //µ±pÖ»ÓÐÒ»¸öº¢×Ó»òÕßÊÇÒ¶½ÚµãÊ±  binaryTreeNode< Pair<K,E> > \*c;  if(p->left!=NULL) //ÈÃcÖ¸Ïòp·Ç¿ÕµÄº¢×Ó  c=p->left;  else  c=p->right;    if(p==BT::root) c=BT::root;  else  {//ÕÒ³öpÊÇppµÄÄÄ¸öº¢×Ó  if(p==pp->left)  pp->left=c;  if(p==pp->right)  pp->right=c;  }  delete p;  BT::treeSize--;  }  void ascend()  {//°ÑËùÓÐ¹Ø¼ü×ÖÉýÐòÊä³ö  BT::inOrder();  }    };  pair.h  #pragma once  #include<iostream>  using namespace std;  //Ä£°åÀà×÷ÎªÄ£°åÀàµÄ³ÉÔ±±äÁ¿ºÃÏñ¾ÍÓÐÎÊÌâ£¿  template<class K,class E>  class Pair  {  public:  K first; //¹Ø¼ü×Ö  E second; //Öµ  int leftSize; //Ë÷Òý¶þ²æËÑË÷Ê÷ÓÃµÄ×ó×ÓÊ÷µÄÔªËØÊý  Pair()  {  first=0; second=0; leftSize=0;  }  Pair(K fir,E sec)  {  first=fir; second=sec; leftSize=0;  }  Pair(const Pair<K,E> &x)  {  first=x.first; second=x.second; leftSize=x.leftSize;  }  void operator=(const Pair<K,E> &x)  {  first=x.first; second=x.second; leftSize=x.leftSize;  }  friend ostream& operator << (ostream& out,Pair<K,E> Pair)  {  out<<"Key="<<Pair.first<<" value="<<Pair.second<<" leftSize="<<Pair.leftSize;  return out;  }  };  二叉树.h  #pragma once  #include<iostream>  #include"arrayQueue.h"  #include<string.h>  using namespace std;  template<class T>  class binaryTreeNode  {  public:  T element; //½ÚµãÖÐµÄÔªËØ  binaryTreeNode<T> \*left,\*right; //×óÓÒ×ÓÊ÷    binaryTreeNode() //ÎÞ²ÎÊý  { left=right=NULL; }  binaryTreeNode(const T& ele)  {//Ö»ÓÐÊý¾Ý²ÎÊý  element=ele;  left=right=NULL;  }  binaryTreeNode(const T& ele,binaryTreeNode\* theLeft,binaryTreeNode\* theRight)  {//Êý¾Ý¼ÓÖ¸Õë  element=ele;  left=theLeft;  right=theRight;  }  };  template<class T>  class linkedBinaryTree  {  public:  linkedBinaryTree()  {  root=NULL;  treeSize=0;  }  ~linkedBinaryTree()  {  //erase();  }  bool empty() const  {  return treeSize;  }  void preOrder()  {  preOrder(root); //ÕâÀïµ÷ÓÃµÄÊÇ¾²Ì¬³ÉÔ±º¯Êý  }  void inOrder()  {  inOrder(root);  }  void postOrder()  {  postOrder(root);  }  void makeTree(const T& element,linkedBinaryTree<T>& left,linkedBinaryTree<T> &right)  {//½«left,rightºÍelementºÏ²¢³ÉÒ»¿ÅÐÂÊ÷  //left£¬rightºÍthis±ØÐë²»Í¬£¡£¡    root=new binaryTreeNode<T>(element,left.root,right.root);  treeSize=left.treeSize+right.treeSize+1;    //×èÖ¹·ÃÎÊleftºÍright  left.root=right.root=NULL;  left.treeSize=right.treeSize=0;  }  binaryTreeNode<T> \*makeTree(char \*pre,char \*in,const int n)  {  root=Sub\_makeTree(pre,in,n);  // cout<<"this is "<<root<<endl;  }  int height()  {  return Sub\_height(root);  }  int size()  {  //return treeSize;  return size(root);  }  int size(binaryTreeNode<T> \*t)  {//·µ»Ø¸ùÎª\*tµÄÊ÷µÄ½ÚµãÊý¡£¶ÔÄ³¸ùµãÇóËü×óÓÒ×ÓÊ÷µÄÔªËØÊý£¬Ïà¼ÓÔÙ¼ÓÉÏËü±¾Éí¼´Îª×Ü½ÚµãÊý¡£  if(t==NULL) return 0;  else  {  int ls=size(t->left); //×ó×ÓÊ÷ÔªËØÊý  int rs=size(t->right); //ÓÒ×ÓÊ÷ÔªËØÊý  return ls+rs+1; //¼ÓÉÏ¸ù½Úµã  }  //¸´ÔÓ¶ÈO(n)  }  void levelOrder()  {  Sub\_levelOrder(root);  }  int max\_width(binaryTreeNode<T> \*t)  {//Çó¶þ²æÊ÷µÄÃ¿²ã×î´ó½ÚµãÊý  arrayQueue< binaryTreeNode<T>\* > q;  binaryTreeNode<T> \*last=root; //lastÖ¸ÏòÃ¿²ã×îÓÒ¶ËµÄÔªËØ  int level=1; //²ãÊý  int width[20]; //±£´æÃ¿²ã½ÚµãÊý  while(t!=NULL)  {  if(t->left!=NULL)  q.push(t->left.element);  if(t->right!=NULL)  q.push(t->right.element);    //ÏÂÒ»¸öt  if(q.empty()==true) break; //¶ÓÁÐÎª¿Õ£¬½áÊøÑ­»·  t=q.front();  q.pop();  if(t==last)  {//ÒÑ¾­É¨Íêµ±Ç°²ã,´ËÊ±¶ÓÁÐÀïÈ«ÊÇÏÂÒ»²ãµÄ½Úµã  width[++level]=q.size();  last=q.back(); //¸üÐÂlast  }  }  //´Ó²ãÊý×éÖÐÕÒ³ö×î´ó½ÚµãÊý  int max=0;  for(int i=1;i<level;i++)  {  if(width[i]>max)  max=width[i];  }    return max;  }  static bool compare(binaryTreeNode<T> \*root1,binaryTreeNode<T> \*root2)  {//ÏÈÅÐ¶ÏÁ½¿ÃÊ÷¸ù½ÚµãÊÇ·ñÓÐ¿Õ£¬Èô¾ù²»Îª¿Õ¾Í±È½ÏÆäÄÚµÄÔªËØÖµ  //ÔÙÅÐ¶Ï×óÓÒ×ÓÊ÷ ------O(N)  if( (root1==NULL&&root2!=NULL) || (root1!=NULL&&root2==NULL) )  return false; //Ê÷µÄ½á¹¹²»Ò»Ñù  else  {  if(root1==NULL && root2==NULL) return true; //¸ù½Úµã¾ùÎª¿Õ£¬µÝ¹éµ½ÖÕµã  if( \*(root1) != \*(root2) ) return false; //¸ù½Úµã¾ù²»Îª¿ÕÇÒÖµ²»Í¬  else  {//¸ù½Úµã²»Îª¿ÕÇÒÖµÏàÍ¬£¬±È½Ï×óÓÒ×ÓÊ÷  bool lt=compare(root1->left,root2->left);  bool rt=compare(root1->right,root2->right);  return lt&&rt;  }  }  }  static void changeIntoPost(T \*pre,T \*in,T \*post,const int n)  {  if(n<1) return;  T root=pre[0];  post[n-1]=root;  T \*p;  int mid; //midÊÇrootÔÚinÖÐµÄË÷Òý    for(p=in;p<in+n;p++)  {  if( \*(p)==root )  break;  }  mid=p-in;  changeIntoPost(pre+1,in,post,mid);  changeIntoPost(pre+mid+1,p+1,post+mid,n-1-mid);  }    protected:  binaryTreeNode<T> \*root; //Ö¸Ïò¸ù½ÚµãµÄÖ¸Õë  int treeSize; //Ê÷µÄ½ÚµãÊý  static void visit(binaryTreeNode<T> \*X)  {//·ÃÎÊº¯Êý  cout<< X->element<<endl;  }  static void preOrder(binaryTreeNode<T> \*t)  {//Ç°Ðò±éÀú  if(t!=NULL)  {  visit(t);  preOrder(t->left);  preOrder(t->right);  }  }  static void inOrder(binaryTreeNode<T> \*t)  {//ÖÐÐò±éÀú  if(t!=NULL)  {  inOrder(t->left);  visit(t);  inOrder(t->right);  }  }  static void postOrder(binaryTreeNode<T> \*t)  {//ºóÐò±éÀú  if(t!=NULL)  {  postOrder(t->left);  postOrder(t->right);  visit(t);  }  }  static void dispose(binaryTreeNode<T> \*t)  {  delete t;  }  int Sub\_height(binaryTreeNode<T> \*t)  {//·µ»Ø¸ùÎª\*tµÄÊ÷µÄ¸ß¶È¡£¶Ô¸ù½ÚµãÇóËüµÄ×óÓÒ×ÓÊ÷µÄ¸ß¶È£¬È¡´óÕß×÷Îª×ÜÊ÷µÄ¸ß¶È  if(t==NULL)  return 0;  else  {  int hl=Sub\_height(t->left);  int hr=Sub\_height(t->right);  if(hl>hr)  return ++hl;  else  return ++hr;  }  //¸´ÔÓ¶ÈO(n)£¬Ã¿Ò»¸öÔªËØ¶¼±È½ÏÁËÒ»±é  }  binaryTreeNode<T> \*Sub\_makeTree(char \*pre,char \*in,const int n)  {//ÓÉÇ°ÐòºÍÖÐÐòÐòÁÐ¹¹ÔìÒ»¿Å¶þ²æÊ÷£¬nÊÇ¶þ²æÊ÷µÄ½Úµã×ÜÊý  if(n<1) return NULL;  char Root=pre[0]; cout<<"Root="<<Root<<"\t";  binaryTreeNode<T> \*b=new binaryTreeNode<T>(Root);  treeSize++;    char \*p; int mid=0; //¸ù½ÚµãÔÚinÖÐµÄË÷Òý  for(p=in;p<in+n;p++)  {//ÔÚÖÐÐòÐòÁÐÖÐÕÒµ½¸ù½Úµã  if( (\*p)==Root )  break;  mid++;  }  cout<<"mid="<<mid<<endl;  b->left=Sub\_makeTree(pre+1,in,mid); //¶Ô×ó×ÓÊ÷µÝ¹é  b->right=Sub\_makeTree(pre+1+mid,in+mid+1,n-mid-1); //¶ÔÓÒ×ÓÊ÷µÝ¹é    return b;  }  void Sub\_levelOrder(binaryTreeNode<T> \*t)  {//²ã´Î±éÀú¶þ²æÊ÷\*t  arrayQueue< binaryTreeNode<T>\* > q; //ÕâÀïÏÈ½øÏÈ³öËùÒÔÓÃ¶ÓÁÐ  while(t!=NULL)  {  visit(t); //·ÃÎÊ t    //½«tµÄº¢×Ó²åÈë¶ÓÁÐ£¬ÏÈ×óºóÓÒ  if(t->left!=NULL)  q.push(t->left);  if(t->right!=NULL)  q.push(t->right);    //·ÃÎÊÏÂÒ»¸ö½Úµã  if(q.empty()==true) return; //ÒÑ·ÃÎÊÍêËùÓÐÔªËØ  t=q.front();  q.pop();  }  }    };  arrayQueue.h  #pragma once  #include<iostream>  #include<cstdlib>  #include<algorithm>  template<class T>  class arrayQueue  {  public:  arrayQueue(int initialCapacity = 10)  {//¹¹Ôìº¯Êý  if(initialCapacity<0) std::cerr<<"¶ÓÁÐ³¤¶È±ØÐë´óÓÚ0£¡"<<std::endl;  else{  Queue=new T[initialCapacity];  arrayLength=initialCapacity;  qFront=qBack=0; //ÕâÀïÊÇ´ÓQueue[1]¿ªÊ¼²åÈëÔªËØ  }  }  ~arrayQueue() {delete [] Queue;}  bool empty() const  {  if(qFront==qBack) return true;  else return false;  }  int size() const  {  return (arrayLength+qBack-qFront)%arrayLength;  }  T& front()  {  if(empty()!=true)  return Queue[(qFront+1)%arrayLength];  else  { std::cerr<<"¶ÓÁÐÎª¿Õ"<<std::endl; exit(1); }  }  T& back()  {  if(empty()!=true)  return Queue[qBack];  else  { std::cerr<<"¶ÓÁÐÎª¿Õ"<<std::endl; exit(1); }  }  T pop()  {//´Ó¶ÓÊ×É¾³ýÔªËØ  T \*p=&front(); //ÕâÀïÒÑ¾­ÅÐ¶ÏÁË¶ÓÁÐÊÇ·ñÎª¿Õ  T temp=(\*p);  qFront=(qFront+1)%arrayLength;  (\*p).~T(); //Îö¹¹Ê×ÔªËØºÃÏñ²»ÄÜ±íÊ¾intµÄÉ¾³ý...£¬¾ÍÊÇÎÞ·¨»Ö¸´µ½³õÊ¼»¯ÒÔÇ°µÄ×´Ì¬  return temp;  }  void push(const T& ele)  {//´Ó¶ÓÎ²Ìí¼ÓÔªËØ  if( (qBack+1)%arrayLength==qFront )  {//¶ÓÁÐ½«Âú£¬¼Ó±¶Êý×é³¤¶È  T \*newQueue=new T[2\*arrayLength];  int start=(qFront+1)%arrayLength;  if(start==0||start==1)  {//Î´ÐÎ³É»·  std::copy(Queue+start,Queue+qBack+1,newQueue);  }  else  {//ÐÎ³ÉÁË»·  std::copy(Queue+start,Queue+arrayLength,newQueue);  //¸´ÖÆµÚ2¶Î(start,¶ÓÁÐÄ©¶Ë£¬ÐÂ¶ÓÁÐÆðµã£©  std::copy(Queue,Queue+qBack+1,newQueue+(arrayLength-start));  //¸´ÖÆµÚ1¶Î£¨Ô­¶ÓÁÐÊ×¶Ë£¬qback,ÐÂ¶ÓÁÐµÚarraylength-start¸öÎ»ÖÃ£©  }  qFront=(arrayLength)\*2-1;  qBack=arrayLength-1-1; //ÖØÐÂÉèÖÃÊ×Î²ÓÎ±ê  arrayLength=arrayLength\*2;  delete [] Queue;  Queue=newQueue;  }  //°ÑÔªËØ²åÈë¶ÓÁÐµÄÎ²²¿  qBack=(qBack+1)%arrayLength;  Queue[qBack]=ele;    }  void output()  {  for(int i=qFront;i<qBack;i++)  std::cout<<Queue[i];  std::cout<<std::endl;  }  private:  int qFront; //¶ÓÁÐÖÐµÚÒ»¸öÔªËØµÄÇ°Ò»¸öÎ´Öª  int qBack; //¶ÓÁÐ×îºóÒ»¸öÔªËØµÄÎ»ÖÃ  int arrayLength; //¶ÓÁÐµÄÈÝÁ¿  T \*Queue; //¶ÓÁÐÔªËØ  }; | | | |