文章编号:1000 - 582X(2004)01 - 0058 - 04

树结构分级分类存储及其在工程结构 CAD 中的应用*

袁政强[1], 李 宾[2], 谢 晟[2]
(1. 重庆大学 土木工程学院重庆 400030;2. 重庆大学 计算机学院,重庆 400030)

摘 要:传统的树结构数据存储是按元素的大小关系,而对于工程结构 CAD 中的图元不好定义大小关系,给树结构应用带来不便。因此改变树结构的存储方式,将树结构按分级分类方式存储,并给出了一种图形元素的分级等价关系,用等价关系对图元进行分类。采用这种存储方式,能方便地提取具有某一特征的子结构树、删除树结点、分级分类插入树结点等操作。给出的算例表明分级分类的存储能提高搜索速度、方便进行分级分类的子结构树的数据处理,这种树结构存储方式适用于定义有分级等价关系的数据。

关键词:数据结构; 树结构存储; 工程结构 CAD; 等价关系
中图分类号:O153;TP311.12;TB237 **文献标识码**:A

树和二叉树都是一种非线形的数据结构,它能很好地描述数据集合,其中又由于二叉树有确定的分支个数,并有非常良好的递归性质,特别适宜于程序设计[1-3]。因此笔者把二叉树作为主要的研究对象。在传统的存储方式中,结点的内容可分成2 部分:结点的数据,指向子结点的指针对;在以往的程序设计中,结点的数据一般都是预定义的数据类型,而在本文应用中,笔者将结点的数据扩展到了模板对象,将对象作为结点的数据可以方便的改变树结构的存储方式。在文献[4]的分类存储中,树结构存储于数据相关。笔者给出的树结构分类存储只与数据的等价关系有关,只要对存储的数据定义分级等价关系,就能分级分类进行存储。

对于每个结点的数据,笔者通过定义分级等价关系来对数据进行分类。对于每个对象应用分级等价关系对它进行判断,将数据插入到树中合适位置。关于数据的等价关系定义,笔者从 2 个方面进行了阐述,从数学[5]角度,给出了它的数学定义和表示;从程序角度,对于对象,定义了两元素为等价的涵义,然后按这种等价关系将对象分成为不同的类。依次类推,可以定义元素在不同级别的等价关系,将对象分层所需要的层次级别。具体实现参看正文中的详细论述。

在笔者的应用中,正是由于采用了这种方式,所以对于结点的插入,选择和删除等操作是非常方便的。在工程结构的应用中,一个图元就可以按照分级分类来对它进行操作:在第一级元素按层放入它所属的等价类,在将它放入此类的下一级等价分类中。选择和删除时,只要按照它的层次和类别要求,分级的查找到满足条件的结点数据即可——按照这种方式查找是非常高效的,不需要遍历整棵树,极大的提高了搜索效率,对于工程 CAD 这种处理大量图元的应用是有很大帮助的[6]。

文中讨论的这种树结构适用于有分级分类等价关系的数据,提到的应用,只是这种树结构在工程 CAD 方面的一个体现,至于其他应用,读者可以根据需要,自行研讨和扩展。

1 等价关系定义和对象分级分类

从数学上看,等价关系是一个对象集合 S 的二元关系。即集合 S 中的两元素之间,有等价和不等价两种状态。假定我们用符号“ ~ ”表示集合 S 上的某种关系,这种关系对于集合 S 中的任意对象 x,y,z,下列性质成立:

1) 对于任意对象 x,$x \sim x$(即等价于自身),称 ~ 具有自反性质(reflexive);

2) 对于任意对象 x 和 y,如果 $x \sim y$,则 $y \sim x$,称 ~ 具有自对称性质(symmetric);

3) 对于任意对象 x,y 和 z,如果 $x \sim y$,$y \sim z$,则 $x \sim z$,称 ~ 具有自传递性质(transitive);

满足以上性质的关系称为等价关系。

应用 S 上的等价关系可以对集合 S 进行分类,即

* 收稿日期:2003 - 07 - 28
基金项目:基金项目:国家自然科学基金(59378351)
作者简介:袁政强(1962 -),男,贵州贵阳人,重庆大学副研究员,主要从事工程力学计算和钢筋混凝土结构研究。

用等价关系可定义新的集合 R，R 中的每个元素是集合 S 的子集合。这些子集合有以下性质：

1）根据自反性质，S 的任一元素属于 R 的某一元素；

2）根据自对称性质，S 中等价的两元素属于 R 的同一元素子集合；

3）根据自传递性质，R 的任意 2 个元素子集合不相交；

一个集合上可以定义多种等价关系，例如，在整数的集合中，两数相等的关系是等价关系、两数被 2 除余数相等的关系，两数被 3 除余数相等的关系等等。如两数相等关系将数分类为与集合个数一样多的类，两数被 2 除余数相等的关系将数分类为奇数和偶数两类。

对于 C++ 中的对象[3-4]，我们可以按要求定义对象间的等价关系。例如对于以下有 4 个成员数据的类，可以按第一个成员的相等关系来定义对象的等价关系：

```
class Element{
private:
  int a, b, c, d;
public:
  int IsEqual(const Elements &ob){return a == ob.a;}
  }
```

对于以上定义的对象，就定义了满足的一个成员元素为等价的等价关系。即按这种等价关系将以上类分成为不同的之类。

如果在 a 相等的同类元素中需要再次进行分类，我们还可以定义等价关系，比如按第 2 个元素相等分类，有：

```
class Element{
private:
  int a, b, c, d;
public:
  int IsEqualA(const Elements &ob){return a == ob.a;}
  int IsEqualB(const Elements &ob){return b == ob.b;}
  }
```

同理，如果在 a 和 b 相等的同类元素中需要再次进行分类，我们还可以定义等价关系，比如按第 3 个元素相等分类，这样，元素 a、b 和 c 就是不同的级。IsSame 函数是判断从 levels 级到 level 级是否全等价。我们把以上类的等价关系写成带级的等价函数为：

```
class Element{
private:
  int a, b, c, d;
public:
  Element(int ca = 0, int cb = 0, int cc = 0, int cd = 0){
    a = ca;b = cb;c = cc;d = cd;}
  int IsEqual(const Elements &ob,int level){
  switch(level){
    case 1:return a == ob.a;
    case 2:return b == ob.b;
    case 3:return c == ob.c;
    case 4:return d == ob.d;}
  }
  int IsSame(const Element &ob, int level, int levels = 1){
    int i = levels; while(i <= level) if(!IsEqual(ob,i++)) return 0;
    return 1;}
  }
```

2　树结构的分级分类存储

树结构的表示方法很多，最常用的有广义表表示、双亲表示和左子女右兄弟二叉树表示 3 种。在 C++ 用二叉树表示中，又分为在分支结点放置数据和不放数据两种。笔者采用在每一个结点都放置数据的存储方式。树结构的模板节点类定义如下：

```
template < class T > class Tree;
template < class T > class TreeNode{
friend class Tree;
private:
  Tree Node < T > *left, *right;
  T data;
public:
  Tree Node():left(NULL),right(NULL){};
  Tree Node(T item,TreeNode < T > *cleft = NULL,
  Tree Node < T > *cright = NULL):data(item),
    left(cleft),right(cright){};
  T GetData(){return data;};
  Tree Node < T > *GetLeft()const{return left;}
  TreeNode < T > *GetRight()const{return right;}
  void SetData(T item){data = item;}
  void SetLeft(TreeNode < T > *L){left = L;}
  void SetRight(TreeNode < T > *R){right = R;}
  };
```

在下面的二叉树定义中，只给出了应用等价关系进行元素增加的函数，其他函数与一般数据结构书上的没有太大区别而省去了。

```
template < class T > class Tree{
public:
  Tree():root(NULL){}
  Tree(T item){
    root = new TreeNode < T >;
```

```
    root -> SetData(item);}
  virtual ~ Tree(){destroy(root);}
  void Add(T &ob); // 自加函数
  Tree Node <T> * Select(T &ob, int &TopNum);
private:
  Tree Node <T> *root; // 根结点
  Tree Node <T> * SelectSet[20]; // 子树结构
  int top, level;
  void Add(T &ob,TreeNode <T> * tempRoot,int depth);
  void Select(T &ob,TreeNode <T> * tempRoot, int depth);
  void InitSelect();
};
```

Add(T &ob,TreeNode < T > * tempRoot,int depth) 函数是在 tempRoot 树中按 depth 级的等价关系放入 ob 数据,在以下函数的第三句中,判断放入数据与 tempRoot 的树是否按 depth 级等价,如果等价把数据放入 tempRoot 的左子女数中,判断级别加 1;如果不等价把数据放入 tempRoot 的右兄弟数中,判断级别不增加。

```
template < class T > void Tree < T >::Add(T &ob,TreeNode < T > * tempRoot,int depth){
  if(tempRoot == NULL) {tempRoot = new TreeNode < T > (ob); return;}
  if(ob -> IsEqual(tempRoot -> GetData(),depth)){
  // 等价时加在 left 指针位置
  if(tempRoot -> left == NULL){
  Tree Node <T> *q = new TreeNode <T> (ob);
  tempRoot -> left = q; return;}
  Add(ob,tempRoot -> left, ++ depth);}
  else{
  // 非等价时加在 right 指针位置
  if(tempRoot -> right == NULL){
  Tree Node <T> *q = new TreeNode <T> (ob);
  tempRoot -> right = q; return;}
  Add(ob,tempRoot -> right,depth); }
  }
```

把一个数据加入一个数中时,只要将数据按级别 1 加入根树中即可,在以上的 Add 函数中一判断了根树为 NULL 的情况。加入后的结果树结构如图 1。函数如下:

```
template < class T > void Tree < T >::Add(T &ob){Add(ob,root,1);}
```

按等价关系分类存储的目的是要能方便地从树结构中选取子树。如果要从树结构中选取满足两级等价关系的子树,首先要构造目标数据。例如要选取满足 Element 类中 a 为 3、b 为 5 的子树,目标数据是 Element ob(3,5); ob 的 c、d 取缺省,在调用时给出选择集的级

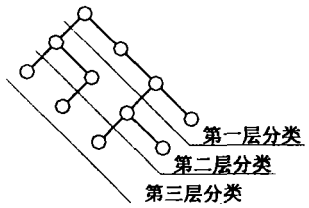

图 1 树结构图

别为 2。选择集的结果放在树结构的 2 个数据域中,SelecetSet 数组和 top。Select 数组中从 0 到 top - 1 是满足等价条件的分支结点,Select[top] 使满足等价条件的子树,即子数中的元素都满足条件。得到的子树结构如图 2。下面的私有选择函数是从 tempRoot 按 depth 级别选择子结点和子树,选择的最大级别是 level。

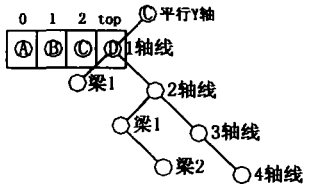

图 2 选择树结构图

```
template < class T >
void Tree < T >::Select(T &ob,TreeNode < T > * tempRoot,int depth){
  if(depth > level) return;
  if(ob -> IsEqual(tempRoot -> GetData(),depth)){
  if(IsSame(ob,level)) SelectSet[top ++] = temp Root;
  if(tempRoot -> left == NULL) return;
  Select(ob,tempRoot -> left, ++ depth);}
    else{
  if(tempRoot -> right == NULL) return;
  Select(ob,tempRoot -> right,depth);
    }
  };
```

在树中的选择函数如下,ob 为选择目标对象,TopNum 调用前是控制级别,返回后是 top 值。

```
template < class T > TreeNode < T > * Tree < T >::Select(T &ob, int &TopNum){
    InitGet(); level = TopNum;
    Select(ob,root,1);
    TopNum = top;
    return SelectSet[0];
  }
```

3 树结构分级分类存储在工程结构 CAD 中的

工程结构 CAD 的图元由轴线、梁、柱、剪力墙等构成,几何特征数据是点。可以把图元抽象为实体,几何特征点和抽象实体的定义如下:

```
class Point{
```

图 3　工程结构 CAD 树存储结构图

```
public:
  long int x, y, z;
}
class CEntity{
public:
  intm_floor;// 图元层次信息
  intm_type;// 图元类型(EEntityType)
  Point m_start, m_end;
public:
  int IsEqual(CEntity * ob, int depth){
    switch(depth){
    // 按楼层分类
  case 1: return m_floor = = ob - > m_floor;
// 按轴线、梁、柱分类
    case 2: return m_type = = ob - > m_type;
// 按平行于 Y 轴与否分类
    case 3: return (m_start.x = = m_end.x)
= = (ob - > m_start.x = = ob - > m_end.x);
// 按同轴线梁分类
  case 4: returnm_start.x = = ob - > m_star.x;
// 按平行于 X 轴与否分类
  case 5: return (m_start.y = = m_end.y) = =
(ob - > m_start.y = = ob - > m_end.y);
// 按同轴线梁分类
  case 6: returnm_start.y = = ob - > m_star.y;
    }
  };
};
```

实体的等价分类关系是按工程结构 CAD 中常用的处理关系分类。各级分类在程序中已说明。

选择对象可以这样构造，如果要取出第 2 层平行于 Y 轴的全部梁，选择对象的构造为：

ob. m_floor = 2; ob. m_type = 2; ob. m_start. x = 300; ob. m_end. x = 300;

level 取 3 即可；如果要取出第 2 层平行于 Y 轴的 x 坐标为 300 的全部梁，用以上对象，level 取 4。

树结构中的操作与存储数据的关系要尽量分离。笔者给出的树结构，操作只用到了数据的等价关系性质，这为存储一般数据给出了有利条件。只要把要存储的数据赋予等价关系，就能进行分类存储。

参考文献：

[1] 殷人昆. 数据结构 - 用面向对象方法与 C + + 描述[M]. 北京：清华大学出版社，1999.
[2] 傅清祥，王晓东. 算法与数据结构[M]. 北京：电子工业出版社，1998.
[3] 蔡子经，施伯乐. 数据结构教程[M]. 上海：复旦大学出版社，1984.
[4] 芦苇，杨晖. 分类树算法及其在通用档案管理系统中的应用[J]. 重庆大学学报，2002，25(4)：72 - 75.
[5] 谢邦杰. 抽象代数[M]. 上海：上海科技出版社，1982.
[6] 李于剑. Visual C + + 实践与提高 - 图形图像编程篇[M]. 北京：中国铁道出版社，2001.

The Storage of Tree Data Structure by Gradation and Classification and Application in CAD for Engineering Structure

YUAN Zheng-qiang, LI Bin, XIE Sheng
(1. College of Civil Engineering, Chongqing University, Chongqing 400030, China;
2. College of Computer, Chongqing University, Chongqing 400030, China)

Abstract: The traditional storage of tree data structure is according to the relation of the elements' frant-and-back. Since it is difficult to define the frant-and-back relation of graphics units in CAD for Engineering structure, the application of the tree structure becomes inconvenient. The authors change the storing method for the tree structure into gradation and classification. And a graded equivalence relation of graphics units is brought forward, which grade the graphics units with the equivalence relation. By using this storing method, we can easily accomplish some operations, such as picking up the subtree with a certain character, deleting a node, insert a graded and classified node, etc. The example provided indicates that the storage by gradation and classification can speed up searching and process the data of the subtree by gradation and classification expediently. This kind of storing method for the tree structure is applicable for data which contain the graded equivalence relation.

Key words: data structure; storage of data structure; CAD for engineering structure; equivalence relation

(编辑　姚　飞)