数据结构作业(第九次)

PB20111686 黄瑞轩

6.73

这里默认每个字符代表一个结点，结点类型如下：

```
constexpr null = -1;
class BNode {
public:
    Elem val = null;
    BNode* FirstChild = nullptr;
    BNode* NextSibling = nullptr;
    BNode(Elem val) {
        this->val = val;
    }
}
```

这里默认子表的父亲一定是一个结点，即不会出现(A)类似的结构。因为要求构造树，所以默认最高层次只有一个结点。算法如下：

```
string s;// 提供的字符序列
int p = 0;// 当前读到s的第几个元素
bool IsAlpha(char ch) {
    return ch <= 'Z' && ch >= 'A';
}
void MakeCSTByString(BNode* T) {
    int sn = s.size();
    for (; p < sn; p++) {
        if (IsAlpha(s[i])) {
            //新建节点
            T = new BNode(s[i]);
        }
        else if (s[i] == '(') {
            //代表进入下一层
            MakeCSTByString(T->FirstChild);
        }
        else if (s[i] == ',') {
            //代表进入下一邻居
            MakeCSTByString(T->NextSibling);
        }
        break;
    }
    return;
}
```

7.16

```
class Vertex {
public:
    //这里用vertexNum表示顶点的特征
    int vertexNum = -1;
    vector<int> neighbors;
    Vertex(int num) {
        this->vertexNum = num;
    }
    bool SearchNeighbor(int v) {
        //查看邻居中是否有编号为v的
        int size = neighbors.size();
        for (int i = 0; i < size; i++) {
            if (neighbors[i] == v) {
                return true;
            }
        }
        return false;
    }
    void DeleteNeighbor(int v) {
        int size = neighbors.size();
        int no = -1;//此时一定能删除
        for (int i = 0; i < size; i++) {
```

```
            if (neighbors[i] == v) {
                no = i;
                break;
            }
        }
        for (int i = no; i < size - 1; i++) {
            neighbors[i] = neighbors[i + 1];
        }
        auto last = neighbors.end();
        last--;
        neighbors.erase(last);
    }
}
class Graph {
public:
    vector<Vertex> V;
    enum kindEnum { DG, UDG };
    int kind;//图的类型，有向图或无向图
    int SearchVertex(Vertex v) {
        // 搜索有无特征为小v的顶点，有则返回在大V中的序号
        int size = V.size();
        for (int i = 0; i < size; i++) {
            if (v.vertexNum == V[i].vertexNum)return i;
        }
        return -1;
    }
    bool InsertVertex(Vertex v) {
        if (SearchVertex(v) == -1) {
            V.push_back(v);
            return true;
        }
        return false;
        //未成功插入
    }
    bool InsertArc(Vertex v, Vertex w) {
        if (SearchVertex(v) != -1 && SearchVertex(w) != -1) {
            if (kind == DG) {
                // 有向图情况
                if (!v.SearchNeighbor(w.vertexNum)) {
                    // v中没有w作为邻居
                    v.neighbors.push_back(w.vertexNum);
                    return true;
                }
            }
            else {
                // 无向图情况
                if (!v.SearchNeighbor(w.vertexNum)) {
                    // v中没有w作为邻居，那么w肯定也没有v作为邻居
                    v.neighbors.push_back(w.vertexNum);
                    w.neighbors.push_back(v.vertexNum);
                    return true;
                }
            }
        }
        return false;
    }
    bool DeleteVertex(Vertex v) {
        int loc = SearchVertex(v);
        if (loc != -1) {
            if (kind == DG) {
                //有向图，需要遍历所有顶点
                int size = V.size();
                for (int j = 0; j < size; j++) {
                    if (V[j].SearchNeighbor(v.vertexNum)) {
                        V[j].DeleteNeighbor(v.vertexNum);
                    }
                }
            }
            else {
                // 无向图，遍历v的邻居即可
                int size = v.neighbors.size();
                for (int j = 0; j < size; j++) {
                    V[v.neighbors[j]].DeleteNeighbor(v.vertexNum);
                }
            }
            // 删除顶点v
            int size = V.size();
```

```
            for (int j = loc; j < size - 1; j++) {
                V[j] = V[j + 1];
            }
            auto last = V.end();
            last--;
            V.erase(last);
            return true;
        }
        return false;
    }
    bool DeleteArc(Vertex v, Vertex w) {
        if (SearchVertex(v) != -1 && SearchVertex(w) != -1) {
            if (kind == DG) {
                if (V[SearchVertex(v)].SearchNeighbor(w.vertexNum)) {
                    V[SearchVertex(v)].DeleteNeighbor(w.vertexNum);
                }
            }
            else {
                if (V[SearchVertex(v)].SearchNeighbor(w.vertexNum)) {
                    V[SearchVertex(v)].DeleteNeighbor(w.vertexNum);
                }
                else {
                    return false;
                }
                if (V[SearchVertex(w)].DeleteNeighbor(v.vertexNum)) {
                    V[SearchVertex(w)].DeleteNeighbor(v.vertexNum);
                }
                else {
                    return false;
                }
            }
            return true;
        }
        return false;
    }
}
```