广义表深度计算

用代码实现求解广义表的深度，以下广义表为测试案例：

（1）A=（）

（2）B=（a,b）

（3）C=（a,b,(c,d)）

（4）D=（a,b,（c,d）,（e，（f），h））

代码实现：

import java.util.Stack;

/\*\*

\* 广义表操作：

\* 1、广义表的构造 ：

\* 1.1 构造一个空的广义表

\* 1.2 根据现有的广义表，构造一个新的广义表

\* 1.3 根据广义表字符串构造一个广义表

\* 2、广义表的深度

\* 3、广义表的长度

\* 4、按照深度优先顺序打印广义表

\* 5、求广义表表头

\* 6、求广义表表尾

\*

\*/

public class GeneralizedTable {

public static final int TAG\_ITEM = 0; // 原子节点

public static final int TAG\_TABLE = 1; // 表节点

/\*

\* 广义表支持的符号包括'(' , ')' , '{' , '}' , '[' , ']'

\* 广义表表示符号,使用字符串构造广义表时第一个字符必须是'(', '{' , '[' 之一 并以')' , '}' , ']' 之一结束，

\* 并且各符号相对应

\*/

private char mStartSymb = '(';

private char mEndSymb = ')';

private Node mGenTable;

public GeneralizedTable() {

mGenTable = new Node(null, null, TAG\_TABLE, null);

}

// 使用广义表 src 构造一个新的广义表

public GeneralizedTable(GeneralizedTable src) {

if (src != null) {

mGenTable = src.mGenTable;

}

}

/\*\*

\* @param genTable

\*/

public GeneralizedTable(String genTable) {

if (genTable == null) {

throw new NullPointerException(

"genTable is null in constructor GeneralizedTable!...");

}

initTable(genTable);

}

private void initTable(String genTable) {

String ts = genTable.replaceAll("\\s", "");

int len = ts.length();

Stack<Character> symbStack = new Stack<Character>();

Stack<Node> nodeStck = new Stack<Node>();

initSymbolicCharactor(ts);

mGenTable = new Node(null, null, TAG\_TABLE, null);

Node itemNode, tableNode = mGenTable, tmpNode;

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

if (ts.charAt(i) == mStartSymb) {

tmpNode = new Node(null, null, TAG\_TABLE, null);

// tableNode = tableNode.mPt;

symbStack.push(ts.charAt(i));

if (symbStack.size() > 1) {

nodeStck.push(tableNode);

tableNode.mPh = tmpNode;

tableNode = tableNode.mPh;

} else {

tableNode.mPt = tmpNode;

tableNode = tableNode.mPt;

}

} else if (ts.charAt(i) == mEndSymb) {

if (symbStack.isEmpty()) {

throw new IllegalArgumentException(

"IllegalArgumentException in constructor GeneralizedTable!...");

}

if (symbStack.size() > 1) {

tableNode = nodeStck.pop();

}

symbStack.pop();

} else if (ts.charAt(i) == ',') {

tableNode.mPt = new Node(null, null, TAG\_TABLE, null);

tableNode = tableNode.mPt;

} else {

itemNode = new Node(null, null, TAG\_ITEM, ts.charAt(i));

tableNode.mPh = itemNode;

}

}

if (!symbStack.isEmpty()) {

throw new IllegalArgumentException(

"IllegalArgumentException in constructor GeneralizedTable!...");

}

}

private void initSymbolicCharactor(String ts) {

mStartSymb = ts.charAt(0);

switch (mStartSymb) {

case '(':

mEndSymb = ')';

break;

case '{':

mEndSymb = '}';

break;

case '[':

mEndSymb = ']';

break;

default:

throw new IllegalArgumentException(

"IllegalArgumentException ---> initSymbolicCharactor");

}

}

public void print() {

print(mGenTable);

}

private void print(Node node) {

if (node == null) {

return;

}

if (node.mTag == 0) {

System.out.print(node.mData.toString() + " \t");

}

print(node.mPh);

print(node.mPt);

}

public int depth() { // 广义表的深度

if (mGenTable == null) {

throw new NullPointerException("Generalized Table is null !.. ---> method depth");

}

return depth(mGenTable);

}

private int depth(Node node) {

if (node == null || node.mTag == TAG\_ITEM) {

return 0;

}

int depHeader = 0, depTear = 0;

depHeader = 1 + depth(node.mPh);

depTear = depth(node.mPt);

return depHeader > depTear ? depHeader : depTear;

}

public int length() { // 广义表的长度

if (mGenTable == null || mGenTable.mPt == null) {

return -1;

}

int tLen = 0;

Node node = mGenTable;

while (node.mPt != null) {

node = node.mPt;

if (node.mPh == null && node.mPt == null) {

break;

}

tLen++;

}

return tLen;

}

public GeneralizedTable getHeader() {

if (isEmpty())

return null;

Node node = mGenTable.mPt;

GeneralizedTable gt = new GeneralizedTable();

gt.mGenTable.mPt = node.mPh;

return gt;

}

public GeneralizedTable getTear() {

if (isEmpty())

return null;

Node node = mGenTable.mPt;

GeneralizedTable gt = new GeneralizedTable();

gt.mGenTable.mPt = node.mPt;

return gt;

}

public boolean isEmpty() {

if (mGenTable == null) {

return true;

}

Node node = mGenTable.mPt;

return node == null || node.mPh == null;

}

public class Node {// 广义表节点

Node mPh; // 广义表的表节点

Node mPt; // 广义表表尾节点

int mTag; // mTag == 0 , 院子节点 ; mTag == 1 , 表节点 。

Object mData; // 广义表的数据值

public Node(Node ph, Node pt, int tag, Object data) {

mPh = ph;

mPt = pt;

mTag = tag;

mData = data;

}

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// String tStr = "(\*\*\*\*)";

String p = "()";

String p2 = "( a,b)";

String p3 = "( a,b,(c,d))";

String p4 = "( a,b,（c,d）,（e，（f），h）)";

// String space = "()";

//String big = "{{a,b},{{a,g},{h},{a,n,f,{a,b,c}}},c}";

//String middle = "[[p],[[d,f,[g]]],[h],[2]]";

GeneralizedTable gTab = new GeneralizedTable(middle);

// GeneralizedTable header, tear;

// // gTab.print();

// // System.out.println();

System.out.println("length: " + gTab.length());

System.out.println("depth: " + gTab.depth());

//

// header = gTab.getHeader();

// if (header != null) {

// System.out.println("header: ");

// header.print();

// }

// tear = gTab.getTear();

//

// if (tear != null) {

// System.out.println("tear: ");

// tear.print();

// }

// gTab.print();

// System.out.println();

// GeneralizedTable gTab4 = null;

// GeneralizedTable gTab2 = new GeneralizedTable(gTab4);

// gTab2.print();

// gTab2 = new GeneralizedTable(gTab);

// gTab2.print();

}

}

思考题

微软面试原题：

（1）给你一幅由 N × N 矩阵表示的图像，其中每个像素的大小为 4 字节。请你设计一种算法，将图像旋转 90 度。

（2）不占用额外内存空间能否做到？

示例 1:

给定 matrix =

[

[1,2,3],

[4,5,6],

[7,8,9]

],

原地旋转输入矩阵，使其变为:

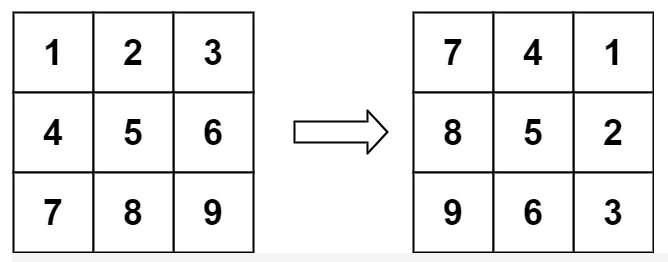
[

[7,4,1],

[8,5,2],

[9,6,3]

]



示例 2:

给定 matrix =

[

[ 5, 1, 9,11],

[ 2, 4, 8,10],

[13, 3, 6, 7],

[15,14,12,16]

],

原地旋转输入矩阵，使其变为:

[

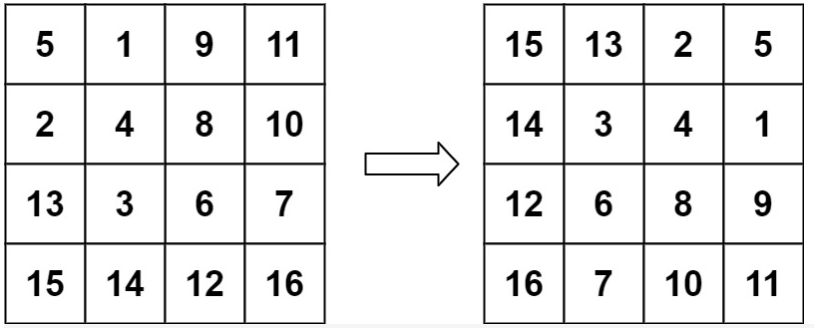
[15,13, 2, 5],

[14, 3, 4, 1],

[12, 6, 8, 9],

[16, 7,10,11]

]



答案解析：

（1）<https://leetcode-cn.com/problems/rotate-matrix-lcci/solution/xuan-zhuan-ju-zhen-by-leetcode-solution/>

（2）<https://leetcode-cn.com/problems/rotate-matrix-lcci/solution/c-tu-jie-yuan-di-cao-zuo-ji-bai-shuang-bai-vv-by-t/>