广义表深度计算

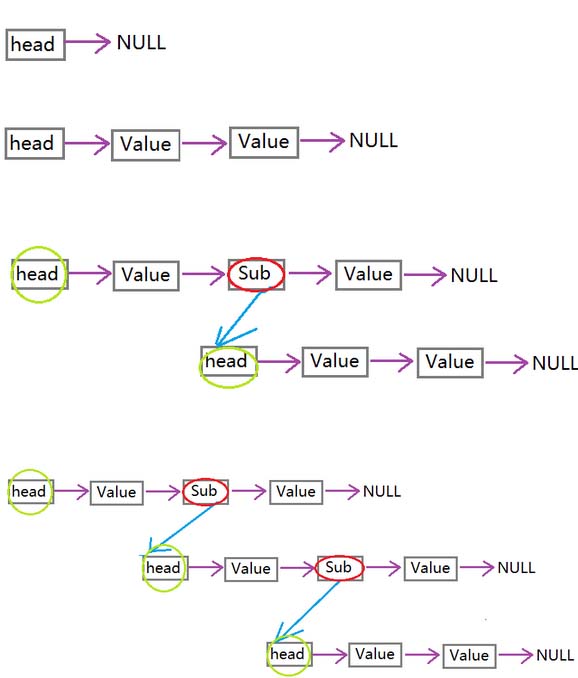
用代码实现求解广义表的深度，以下广义表为测试案例：

（1）A=（）

（2）B=（a,b）

（3）C=（a,b,(c,d)）

（4）D=（a,b,（c,d）,（e，（f），h））



由上图我们可以看到，广义表的节点类型无非head、value、sub三种，这里设置枚举类型，利用枚举变量来记录每个节点的类型：

enum Type

{

HEAD, //头节点

VALUE, //值节点

SUB, //子表节点

};

每个节点都有自己的类型以及next指针，除此之外，如果该节点是VALUE类型还要分配空间存储该节点的有效值；但是若该节点是SUB类型，就需定义一个指针指向子表的头。

这里我们可以用联合来解决这个问题。

（联合（或共同体）是一种不同数据类型成员之间共享存储空间的方法，并且联合体对象在同一时间只能存储一个成员值）

代码实现：

[cpp] view plain copy

#include <iostream>

using namespace std;

//表示广义表的结点类型

enum NodeType

{

HEAD\_TYPE,//头结点类型

VALUE\_TYPE,//值结点类型

SUB\_TYPE//子表类型

};

//表示广义表结点的结构体

struct GeneraListNode

{

NodeType \_type;//结点类型

GeneraListNode \*\_next;//存放结点的下一个元素的地址

//一个结点要么是值结点要么是子表，故用联合体来存放节省一定的空间

//若是值结点则存放的是值，是子表结点的话存放的是子表结点头结点的地址

union{

char \_value;

GeneraListNode \*\_subLink;

};

GeneraListNode(NodeType type = HEAD\_TYPE, char value = '\0')

:\_type(type)

,\_next(NULL)

{

if (type == VALUE\_TYPE)

{

\_value = value;

}else if(type == SUB\_TYPE)

{

\_subLink = NULL;

}

}

};

class GeneraList

{

private:

GeneraListNode \*\_link;//用来存放广义表头结点地址

public:

GeneraList(const char \*str)

:\_link(NULL)

{

\_CreateGeneraList(\_link, str);//根据指定序列创建广义表

}

~GeneraList()

{}

public:

void Print();//对外提供的打印广义表的接口

int Size();//广义表中值结点的数目的对外获取接口

int Depth();//广义表的最深层次的对外获取接口

private:

void \_CreateGeneraList(GeneraListNode \*& link, const char \*& str);

bool \_IsValue(const char ch);//判断指定字符是否为值结点所允许的类型

int \_Size(GeneraListNode \*head);//计算广义表中值结点的数目的实现

int \_Depth(GeneraListNode \*head);//计算广义表的最深层次的实现

void \_Print(GeneraListNode \*link);//打印广义表的接口的底层实现

};

//创建广义表

void GeneraList::\_CreateGeneraList(GeneraListNode \*& link, const char \*& str)

{

//广义表最前端有一个头结点，用来记录实现广义表链表的首地址

//故每次调用该创建广义表的函数首先创建一个头结点

GeneraListNode\* head = new GeneraListNode(HEAD\_TYPE, NULL);

head->\_next = NULL;

link = head;

GeneraListNode\* cur = link;//用来记录创建广义表链表时当前创建出的结点位置游标指针

str++;//将广义表序列后移,相当于跳过了'('

while(\*str != '\0')

{

if(\_IsValue(\*str)){//如果当前扫描到的字符是值

//创建一个值结点

GeneraListNode\* newNode = new GeneraListNode(VALUE\_TYPE, \*str);

newNode->\_next = NULL;

cur->\_next = newNode;//将该值结点加入到链表中

cur = cur->\_next;//游标后移

str++;//将广义表序列后移

}else if(\*str == '('){//如果扫描到'('创建子表结点

GeneraListNode\* subLink = new GeneraListNode(SUB\_TYPE, NULL);

subLink->\_next = NULL;

cur->\_next = subLink;//将子表结点加入到链表中

cur = cur->\_next;

\_CreateGeneraList(cur->\_subLink, str);//递归创建子表

}else if(\*str == ')'){

str++;

return;//若扫描到')'表示广义表创建结束

}else{

str++;//空格等其他无效字符跳过

}

}

}

int GeneraList::Size()

{

return \_Size(\_link);

}

//计算广义表值结点的个数

int GeneraList::\_Size(GeneraListNode \*head)

{

int size = 0;

GeneraListNode \*cur = head;

while(cur != NULL){

if(cur->\_type == VALUE\_TYPE){

++size;//遇到值结点则将size加一

}else if(cur->\_type == SUB\_TYPE){

size += \_Size(cur->\_subLink);//遇到子表进行递归

}

cur = cur->\_next;

}

return size;

}

int GeneraList::Depth()

{

return \_Depth(\_link);

}

int GeneraList::\_Depth(GeneraListNode \*head)

{

int depth = 1,maxDepth = 1;//depth表示当前表的深度，maxDepth表示目前最大的深度

GeneraListNode \*cur = head;

while(cur != NULL){

if(cur->\_type == SUB\_TYPE){

depth += \_Depth(cur->\_subLink);

}

if(depth > maxDepth){//更新最大深度

maxDepth = depth;

depth = 1;//将当前深度复位

}

cur = cur->\_next;

}

return maxDepth;

}

void GeneraList::Print()

{

\_Print(\_link);

cout<<endl;

}

//打印广义表

void GeneraList::\_Print(GeneraListNode \*link)

{

GeneraListNode \*cur = link;//遍历广义表的游标

while(cur != NULL){

if(cur->\_type == VALUE\_TYPE){

cout<<cur->\_value;

if(cur->\_next != NULL)

{

cout<<',';

}

}else if(cur->\_type == HEAD\_TYPE){

cout<<"(";

}else if(cur->\_type == SUB\_TYPE){

\_Print(cur->\_subLink);//遇到子表递归打印

if(cur->\_next != NULL)//如果打印完子表后广义表未结束则打印','

{

cout<<",";

}

}

cur = cur->\_next;

}

cout<<")";

}

bool GeneraList::\_IsValue(const char ch)

{

if(ch >= 'a' && ch <= 'z' ||

ch >= 'A' && ch <= 'Z' ||

ch >= '0' && ch <= '(')

{

return true;

}

return false;

}

**测试代码**

[cpp] view plain copy

#include"GeneraList.hpp"

//测试空表

void Test1()

{

GeneraList genList("()");

genList.Print();

cout<<"Size is :"<<genList.Size()<<endl;

cout<<"Depth is :"<<genList.Depth()<<endl<<endl;

}

//测试单层表

void Test2()

{

GeneraList genList("(a,b)");

genList.Print();

cout<<"Size is :"<<genList.Size()<<endl;

cout<<"Depth is :"<<genList.Depth()<<endl<<endl;

}

//测试双层表

void Test3()

{

GeneraList genList("(a,b,(c,d))");

genList.Print();

cout<<"Size is :"<<genList.Size()<<endl;

cout<<"Depth is :"<<genList.Depth()<<endl<<endl;

}

//测试多层表

void Test4()

{

GeneraList genList("(a,b,(c,d),(e,(f),h))");

genList.Print();

cout<<"Size is :"<<genList.Size()<<endl;

cout<<"Depth is :"<<genList.Depth()<<endl<<endl;

}

//测试多层空表

void Test5()

{

GeneraList genList("(((),()),())");

genList.Print();

cout<<"Size is :"<<genList.Size()<<endl;

cout<<"Depth is :"<<genList.Depth()<<endl<<endl;

}

int main()

{

Test1();

Test2();

Test3();

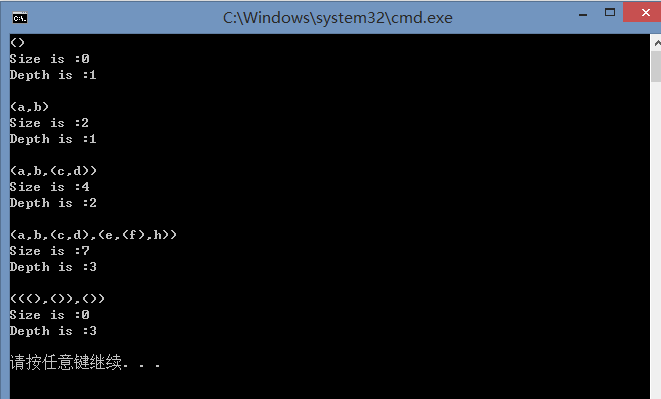
Test4();

Test5();

return 0;

}

运行结果：



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:

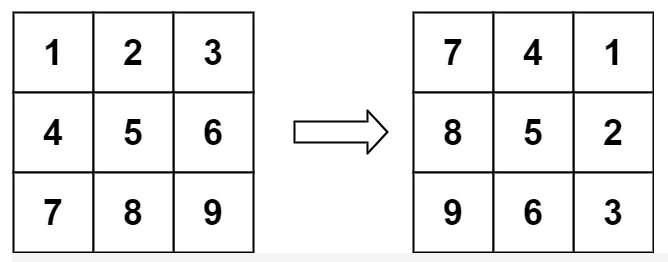
[

[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:

[

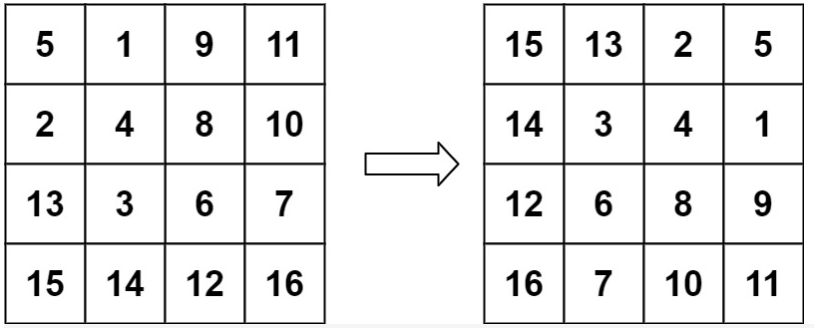
[15,13, 2, 5],

[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/>