1. **一棵满 k 叉树按层次顺序从 1 开始对全部结点编号，若所求结点存在，则：**
2. **各层的结点数目是多少？**
3. **编号为 n 的结点的父结点的编号是多少？**
4. **结点 n 的第 i 个儿子的编号是多少？**
5. **结点 n 有右兄弟的条件是什么？**
6. **第i层：k^(i-1)**
7. **n=1时，父节点不存在；**

**n>1时，对(n-2+k)/k向下取整，即为父节点编号。**

**3、结点n的第k-1个儿子的编号为n\*k。因此，结点n的第i个儿子的编号为n\*k+(i-(k-1))**

**4、(n-1)%k!=0**

1. **已知一棵度为 m 的树中有 ni 个度为 i 的结点，则该树中有多少个叶子结点。**

**i=1,2,3,...,m**

**树的结点个数=n0+n1+n2+...+nm**

**=0\*n0+1\*n1+2\*n2+...+m\*nm+1**

**解方程，可得n0=1+n2+2\*n3+3\*n4+...+(m-1)\*nm**

1. **已知在二叉链表表示的二叉树中，root 为根结点， p↑和 q↑为二叉树中两个结点，试编写算法求距离它们最近的共同祖先。**

#include <iostream>

using namespace std;

#define STACK\_MAXLEN 200

struct Node // 二叉树结点

{

char data;

Node \*lchild, \*rchild;

};

struct Stack // 存放二叉树结点指针

{

Node \*data[STACK\_MAXLEN];

int top;

};

typedef Node \*Btree;

void push(Stack \*s, Node \*value) // 元素入栈

{

s->data[++s->top] = value;

return;

}

Node \*pop(Stack \*s) // 元素出栈并返回

{

return s->data[s->top--];

}

void crtBt(Btree &t) // 读取字符串，先序创建二叉树

{

char c;

cin >> c;

if (c == '/')

t = NULL;

else

{

t = new Node;

t->data = c, t->lchild = NULL, t->rchild = NULL;

crtBt(t->lchild), crtBt(t->rchild);

}

return;

}

void getPath(Btree t, Node \*target, Stack &path) // 获取从根结点到某结点的路径

{

Stack tmp; // 存放未遍历的结点

Node \*cur = t;

// 初始化

path.top = -1, tmp.top = -1;

for (auto &i : path.data)

i = NULL;

for (auto &i : tmp.data)

i = NULL;

// 搜索

while (cur != target)

{

push(&path, cur);

if (cur->lchild) // 有左结点

{

if (cur->rchild) // 有右结点

push(&tmp, cur->rchild);

cur = cur->lchild;

}

else if (cur->rchild) // 没有左结点，有右结点

cur = cur->rchild;

else // 左右结点都没有

{

while (path.data[path.top]->rchild != tmp.data[tmp.top])

pop(&path);

cur = pop(&tmp);

}

}

return;

}

Node \*srhRoot(Btree t, Node \*p, Node \*q) // 搜索pq最近的共同祖先

{

Stack pathP, pathQ;

getPath(t, p, pathP), getPath(t, q, pathQ);

int pos = 0;

while (pathP.data[pos] == pathQ.data[pos])

pos++;

return pathP.data[pos - 1];

}

void deleteBt(Btree t) // 释放二叉树结点

{

if (t == NULL)

return;

deleteBt(t->lchild), deleteBt(t->rchild);

delete t;

}

int main()

{

cout << "创建二叉树，先序遍历，'/'表示孩子为空：";

Btree t;

crtBt(t);

// A

// / \

// B C

// / \ \

// D E F

Node \*p = t->lchild->lchild, \*q = t->lchild->rchild;

Node \*res = srhRoot(t, p, q);

cout << "指针p(D)和指针q(E)的最近的共同祖先为：" << res->data << endl;

deleteBt(t);

return 0;

}

**运行该程序，输出结果如下：**

**创建二叉树，先序遍历，'/'表示孩子为空：ABD//E//C/F//**

**指针p(D)和指针q(E)的最近的共同祖先为：B**

1. **试给出算法求二叉链表表示的二叉树的直径（高度、最大层次数）以及长度等于直径的一条路经（从根到叶子的结点序列）。**

#include <iostream>

using namespace std;

#define STACK\_MAXLEN 200

struct Node // 二叉树结点

{

char data;

Node \*lchild, \*rchild;

};

struct Stack // 存放二叉树结点指针

{

Node \*data[STACK\_MAXLEN];

int top;

};

typedef Node \*Btree;

void initialize(Stack \*s) // 初始化栈

{

for (auto &i : s->data)

i = NULL;

s->top = -1;

return;

}

void push(Stack \*s, Node \*value) // 元素入栈

{

s->data[++s->top] = value;

return;

}

Node \*pop(Stack \*s) // 元素出栈并返回

{

return s->data[s->top--];

}

void crtBt(Btree &t) // 读取字符串，先序创建二叉树

{

char c;

cin >> c;

if (c == '/')

t = NULL;

else

{

t = new Node;

t->data = c, t->lchild = NULL, t->rchild = NULL;

crtBt(t->lchild), crtBt(t->rchild);

}

return;

}

int srhDepth(Btree t, Stack &path) // 求最大层次及其路径，返回最大层次

{

Node \*cur = t;

Stack curPath, tmp;

int maxlen = 0, curlen = 0;

initialize(&path), initialize(&curPath), initialize(&tmp);

while (cur || tmp.top != -1)

{

push(&curPath, cur), curlen++;

if (cur->lchild) // 有左结点

{

if (cur->rchild) // 有右结点

push(&tmp, cur->rchild);

cur = cur->lchild;

}

else if (cur->rchild) // 没有左结点，有右结点

cur = cur->rchild;

else // 左右结点都没有，到叶子结点了

{

if (curlen > maxlen) // 更新最长路径

{

maxlen = curlen;

for (int i = 0; i <= curPath.top; i++)

path.data[i] = curPath.data[i];

path.top = curPath.top;

}

if (tmp.top == -1) // 全部结点都遍历完

break;

else

{

while (curPath.data[curPath.top]->rchild != tmp.data[tmp.top])

pop(&curPath), curlen--;

cur = pop(&tmp);

}

}

}

return maxlen;

}

void printPath(Stack \*path) // 打印一条最长路径

{

for (int i = 0; i <= path->top; i++)

cout << path->data[i]->data << ' ';

return;

}

void deleteBt(Btree t) // 释放二叉树结点

{

if (t == NULL)

return;

deleteBt(t->lchild), deleteBt(t->rchild);

delete t;

}

int main()

{

cout << "创建二叉树，先序遍历，'/'表示孩子为空：";

Btree t;

crtBt(t);

// A

// / \

// B C

// / \ \

// D E F

Stack path;

int maxlen = srhDepth(t, path);

cout << "该二叉树的最大层次数为：" << maxlen << endl

<< "其中一条路径为：";

printPath(&path);

deleteBt(t);

return 0;

}

**运行该程序，输出结果如下：**

**创建二叉树，先序遍历，'/'表示孩子为空：ABD//E//C/F//**

**该二叉树的最大层次数为：3**

**其中一条路径为：A B D**

1. **试给出算法将二叉树表示的表达式按中序遍历输出并加上相应的扩号。**

#include <iostream>

using namespace std;

struct Node // 二叉树结点

{

char data;

Node \*lchild, \*rchild;

};

typedef Node \*Btree;

void crtBt(Btree &t) // 读取字符串，先序创建二叉树

{

char c;

cin >> c;

if (c == '#')

t = NULL;

else

{

t = new Node;

t->data = c, t->lchild = NULL, t->rchild = NULL;

crtBt(t->lchild), crtBt(t->rchild);

}

return;

}

void print(Btree t) // 打印加了括号的表达式

{

Node \*cur = t;

if (cur) // cur不为空

{

if (cur->lchild) // 非叶子结点

{

cout << "(";

print(cur->lchild);

cout << cur->data;

print(cur->rchild);

cout << ")";

}

else // 叶子结点

cout << cur->data;

}

return;

}

void deleteBt(Btree t) // 释放二叉树结点

{

if (t == NULL)

return;

deleteBt(t->lchild), deleteBt(t->rchild);

delete t;

}

int main()

{

cout << "创建二叉树，先序遍历，'#'表示孩子为空：";

Btree t;

crtBt(t);

// 中序：a + b / c - d \* e + f

// \*

// / \

// + +

// / \ / \

// a / e f

// / \

// b -

// / \

// c d

cout << "该中缀表达式带括号输出为：";

print(t);

deleteBt(t);

return 0;

}

**运行该程序，输出结果如下：**

**创建二叉树，先序遍历，'#'表示孩子为空：\*+a##/b##-c##d##+e##f##**

**该中缀表达式带括号输出为：((a+(b/(c-d)))\*(e+f))**