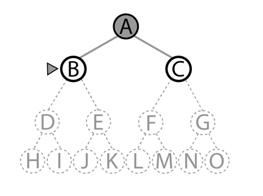
**AI课上代码实现 第一次试验 硬件一班 王倩倩 171491121**

**1 DFS**



一 分析

DFS是先访问根结点，然后遍历左子树接着是遍历右子树

A 是第一个访问的，然后顺序是 B、D，然后是 E。接着再是 C、F、G。在遍历了根结点后，就开始遍历左子树，最后才是右子树。

因此可以借助堆栈的数据结构，由于堆栈是后进先出的顺序，由此可以先将右子树压栈，然后再对左子树压栈，

这样一来，左子树结点就存在了栈顶上，因此某结点的左子树能在它的右子树遍历之前被遍历。

二 实验代码

#include<iostream>

#include <queue>

#include<stack>

using namespace std;

struct Node

{

int nVal;

Node \*pLeft;

Node \*pRight;

Node(int val,Node\* left=NULL,Node \* right=NULL):nVal(val),pLeft(left),pRight(right){}; //构造

};

// 析构

void DestroyTree(Node \*pRoot)

{

if (pRoot==NULL)

return;

Node\* pLeft=pRoot->pLeft;

Node\* pRight=pRoot->pRight;

delete pRoot;

pRoot =NULL;

DestroyTree(pLeft);

DestroyTree(pRight);

}

// 用queue实现的BFS

void BFS(Node \*pRoot)

{

if (pRoot==NULL)

return;

queue<Node\*> Q;

Q.push(pRoot);

while(!Q.empty())

{

Node \*node = Q.front();

cout<<node->nVal<<"->";

if (node->pLeft!=NULL)

{

Q.push(node->pLeft);

}

if (node->pRight!=NULL)

{

Q.push(node->pRight);

}

Q.pop();

}

cout<<endl;

}

// DFS的递归实现

void DFS\_Recursive(Node\* pRoot)

{

if (pRoot==NULL)

return;

cout<<pRoot->nVal<<" ";

if (pRoot->pLeft!=NULL)

DFS\_Recursive(pRoot->pLeft);

if (pRoot->pRight!=NULL)

DFS\_Recursive(pRoot->pRight);

}

// DFS的迭代实现版本（stack）

void DFS\_Iterative(Node\* pRoot)

{

if (pRoot==NULL)

return;

stack<Node\*> S;

S.push(pRoot);

while (!S.empty())

{

Node \*node=S.top();

cout<<node->nVal<<",";

S.pop();

if (node->pRight!=NULL)

{

S.push(node->pRight);

}

if (node->pLeft!=NULL)

{

S.push(node->pLeft);

}

}

}

// 打印树的信息

void PrintTree(Node\* pRoot)

{

if (pRoot==NULL)

return;

cout<<pRoot->nVal<<" ";

if (pRoot->pLeft!=NULL)

{

PrintTree(pRoot->pLeft);

}

if (pRoot->pRight!=NULL)

{

PrintTree(pRoot->pRight);

}

}

int main()

{

Node \*node1=new Node(4);

Node \*node2=new Node(5);

Node \*node3=new Node(6);

Node\* node4=new Node(2,node1,node2);

Node\* node5=new Node(3,node3);

Node\* node6=new Node(1,node4,node5);

Node\* pRoot = node6;

//PrintTree(pRoot);

//DFS\_Recursive(pRoot);

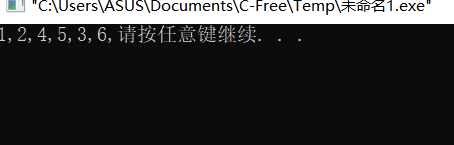
DFS\_Iterative(pRoot);

DestroyTree(pRoot);

return 0;

}

三 实验结果



2 A\*算法

A\*算法是一种静态路网中求解最短路最有效的方法。

公式表示为： f(n)=g(n)+h(n),

其中 f(n) 是从初始点经由节点n到目标点的估价函数，

g(n) 是在状态空间中从初始节点到n节点的实际代价，

(n) 是从n到目标节点最佳路径的估计代价。

保证找到最短路径（最优解的）条件，关键在于估价函数h(n)的选取：

（ 对于连通图）从图中某个顶点v出发，访问此顶点，然后从v的未被访问的邻接点出发深度优先遍历图，直到图中所有和v有路径相通的顶点都被访问到。（对于非连通图），只需要对它的连通分量分别进行深度优先遍历，即在先前一个顶点进行一次深度优先遍历后，若图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作起始点，重复上述过程，直到图中所有的顶点都被访问过为止。