队列：

Package T1;

import java.util.LinkedList;

public class Queue<T> {

//集合存储

LinkedList<T> Q = new LinkedList<T>();

//添加元素到队列末

public void push(T node) {

Q.add(node);

}

//移除节点

public T poll() {

return Q.pollFirst();

}

//判断队列是否为空

public boolean isEmpty() {

return Q.isEmpty();

}

}

满二叉树：

package T1;

public class Tree {

private String mes;

private Tree leftTree;

private Tree rightTree;

//无参构造函数

public Tree() {

mes = "";

leftTree = null;

rightTree = null;

}

//有参构造函数

public Tree(String mes) {

super();

this.mes = mes;

leftTree = null;

rightTree = null;

}

//创建值

public String getCh() {

return mes;

}

//更新值

public void setCh(String mes) {

this.mes = mes;

}

//创建左子树

public Tree getLeftTree() {

return leftTree;

}

//更新左子树

public void setLeftTree(Tree leftTree) {

this.leftTree = leftTree;

}

//创建右子树

public Tree getRightTree() {

return rightTree;

}

//更新右子树

public void setRightTree(Tree rightTree) {

this.rightTree = rightTree;

}

}

类：

package T1;

import T1.Queue;

import T1.Tree;

public class Test1 {

private static char ch = 'A';

private static Queue<Tree> Q;

public static void main(String[] args) {

Tree rootTree = new Tree(ch+"");

//创建一个深度为三的满二叉树

createTree(rootTree,1,3);

//深度优先搜索

dfs(rootTree);

System.out.println();

//宽度优先搜索

bfs(rootTree);

}

//bfs

private static void bfs(Tree rootTree) {

//构造队列

Q = new Queue<Tree>();

//根进队列

Q.push(rootTree);

while(!Q.isEmpty()) {

//得到队首元素

Tree node = Q.poll();

//空元素跳过

if(node == null)continue;

//打印信息

System.out.print(node.getCh());

//左右子树进队列

if(node.getLeftTree()!=null)

Q.push(node.getLeftTree());

if(node.getRightTree()!=null)

Q.push(node.getRightTree());

}

}

//深度优先搜索

private static void dfs(Tree rootTree) {

if(rootTree==null)return ;

System.out.print(rootTree.getCh());

dfs(rootTree.getLeftTree());

dfs(rootTree.getRightTree());

}

//创建一颗满二叉树

//现在层数cur

//最深层数n

private static void createTree(Tree rootTree,int cur,int n) {

//最后一层节点

if(cur == n) return;

rootTree.setLeftTree(new Tree(++ch + ""));

rootTree.setRightTree(new Tree(++ch + ""));

createTree(rootTree.getLeftTree(), cur+1, n);

createTree(rootTree.getRightTree(), cur+1, n);

}

}