C：

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct TNode

{

int key;

struct TNode \*left;

struct TNode \*right;

}TNode, \*Tree;

Tree insert(Tree root, Tree src)

{

if(root == NULL)

{

root = src;

}

else if(src->key > root->key)

{

root->left = insert(root->left, src);

}

else

{

root->right = insert(root->right, src);

}

return root;

}

char l[1000];

#define U 1

#define D 2

#define S ('.')

void print(Tree root, int s, int dir)

{

if(root != NULL)

{

int i;

char buf[10];

sprintf(buf, "|-%d-", root->key);

int len = strlen(buf);

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

{

l[s + i] = S;

}

if(dir == D)

{

l[s] = '|';

}

print(root->left, s + len, U);

l[s] = '\0';

if(root->left == NULL && root->right == NULL)

{

buf[len - 1] = '\0';

printf("%s%s\n", l, buf);

}

else

{

printf("%s%s|\n", l, buf);

}

l[s] = S;

if(dir == U)

{

l[s] = '|';

}

print(root->right, s + len, D);

l[s] = S;

}

}

void printPre(Tree root, int s)

{

if(root != NULL)

{

int i;

char buf[10];

sprintf(buf, "%d-", root->key);

int len = strlen(buf);

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

{

l[s + i] = S;

}

print(root->left, s + len, U);

printf("%s|\n", buf);

print(root->right, s + len, D);

}

}

int main(void)

{

int n;

Tree tree = NULL;

while(scanf("%d", &n) > 0)

{

Tree neo = malloc(sizeof(TNode));

neo->key = n;

neo->left = neo->right = NULL;

tree = insert(tree, neo);

}

printPre(tree, 0);

return 0;

}

C++：

#include <iostream>

using namespace std;

#include <stdio.h>

#include <string.h>

#include <malloc.h>

#include <vector>

#include <algorithm>

#define N 100

typedef struct BiTNode//二叉树结点

{

int data;//数据

struct BiTNode \*lchild,\*rchild;//左右孩子指针

}BiTNode,\*BiTree;

void CreateBalanceTree(BiTree &T,int x)//x 插入的数据

{

if(T == NULL)//若当前树为空

{

T = (BiTree)malloc(sizeof(BiTNode));

T->data = x;

T->lchild = NULL;

T->rchild = NULL;

}

else if(x < T->data)//如果比当前结点小，插入左子树

{

CreateBalanceTree(T->lchild,x);

}

else if(x > T->data)//如果比当前结点大，插入右子树

{

CreateBalanceTree(T->rchild,x);

}

}

vector<string> InOrder(BiTree T,int level,int lr)//右根左的顺序进行遍历,同时计算层次

{

int d,i,len;//

vector<string> v,vr,vl;

string s="";

string pre="";

string t="";

char c[5];

sprintf(c, "%d", T->data);

s=c;

len=s.length()+2;

if(level==1) len-=2;

if(level>1) s="|-"+s;//除了根结点外,左边都加|-

if(T->lchild||T->rchild) s+="-|";//如果结点有孩子

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

{

pre+='.';

}

if(T->rchild != NULL)

{

vr=InOrder(T->rchild,level+1,1);

}

if(T->lchild != NULL)

{

vl=InOrder(T->lchild,level+1,-1);

}

if(lr<0)

{

for(i=0;i<vr.size();i++)

{

vr[i]="|"+pre+vr[i];

}

for(i=0;i<vl.size();i++)

{

vl[i]="."+pre+vl[i];

}

}

if(lr>0)

{

for(i=0;i<vr.size();i++)

{

vr[i]="."+pre+vr[i];

}

for(i=0;i<vl.size();i++)

{

vl[i]="|"+pre+vl[i];

}

}

if(lr==0)

{

for(i=0;i<vr.size();i++)

{

vr[i]="."+pre+vr[i];

}

for(i=0;i<vl.size();i++)

{

vl[i]="."+pre+vl[i];

}

}

v.insert(v.end(),vr.begin(),vr.end());//右

v.insert(v.end(),s);//根

v.insert(v.end(),vl.begin(),vl.end());//左

return v;

}

void print(string s)

{

cout << s<< endl;

}

int main()

{

int n;

char str[500];//输入的字符串

char \*p=NULL;//读入的每个数字

BiTree T = NULL;

vector<string> v;

gets(str);

p= strtok(str, " ");

while ( p != NULL ) //创建二叉树

{

CreateBalanceTree(T,atoi(p));

p = strtok( NULL, " " );

}

v=InOrder(T,1,0);//根的层次为1

for\_each(v.begin(), v.end(), print);

printf("\n");

return 0;

}

Java：

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.StringTokenizer;

public class Main {

static int n;

static int[] a=new int[108];

static int[] left;

static int[] right;

static int[] level;

static int maxl=0;

static int M,N;

static char[][] map;

static boolean[] flag;

static boolean[] flag2;

static void add(int root,int index)

{

if(a[index]<a[root])

if(left[root]==0) { left[root]=index; level[index]=level[root]+1; if(level[index]>maxl) maxl=level[index]; }

else add(left[root],index);

else

if(right[root]==0) { right[root]=index; level[index]=level[root]+1; if(level[index]>maxl) maxl=level[index]; }

else add(right[root],index);

}

static void Fill(int root,int x,int y)

{

int lvl=level[root]+1;

String num=String.valueOf(a[root]);

int len=num.length();

flag[x]=true;

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

{

map[x][y-i]=num.charAt(i);

int h=2<<(maxl-lvl);

for(int j=1;j<h;j++) { map[x+j][y-i]='.'; map[x-j][y-i]='.'; }

}

if(left[root]==0 && right[root]==0 && level[root]<maxl) flag2[x]=true;

if(left[root]!=0)

Fill(left[root],x+(2<<(maxl-lvl))/2,y-8);

if(right[root]!=0)

Fill(right[root],x-(2<<(maxl-lvl))/2,y-8);

}

public static void main(String[] args) throws IOException {

BufferedReader bfr = new BufferedReader(new InputStreamReader(System.in));

StringTokenizer tok=new StringTokenizer(bfr.readLine());

// long begin=System.currentTimeMillis();

int i,j;

for(i=1;tok.hasMoreTokens();i++) a[i]=Integer.parseInt(tok.nextToken());

n=i;

left =new int[n];

right =new int[n];

level=new int[n];

left[0]=1; right[0]=1; level[0]=0; level[1]=1;

for(i=2;i<n;i++)

add(1,i);

M=(2<<(maxl-1))-1;

N=8\*maxl-3;

map=new char[M][N];

flag=new boolean[M];

flag2=new boolean[M];

for(int l=1;l<maxl;l++)

{

int ty=l\*8-1;

int start=(2<<(l-1))-1,d=(2<<l);

for(i=0;i<M;i++) map[i][ty]=map[i][ty-1]=map[i][ty-2]='.';

for(i=start;i<M;i+=d)

{

map[i][ty]='-';

int hei=(2<<(l-1))/2;

for(int ii=0;ii<=hei;ii++)

{

map[i+ii][ty-1]='|';

map[i-ii][ty-1]='|';

}

map[i+hei][ty-2]='-';

map[i-hei][ty-2]='-';

}

}

Fill(1,M/2,N-1);

for(i=0;i<M;i++)

{

if(flag[i])

{

for(j=0;;j++) if(map[i][j]!=0 && map[i][j]!='.') break;

int jj;

for(jj=N-1;jj>j+1;jj--) if(map[i][jj]!=0) System.out.print(map[i][jj]);

if(!flag2[i]) System.out.print(map[i][jj]+""+map[i][jj-1]);

System.out.println();

}

}

// System.out.println(System.currentTimeMillis()-begin);

}

}