Написать программу, находящую наибольшую по числу вершин компоненту связности графа.

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.Scanner;

**public** **class** Comp {

**public** **static** **int** *n*;

**public** **static** **int**[] *use*;

**public** **static** **int**[][] *g*;

**public** **static** **int** *count*;

**public** **static** **int** *mark* = 0;

**public** **static** **void** dfs(**int** v) {

*use*[v] = *mark*;

**for** (**int** i=0;i<*n*;i++) {

**if** (*g*[v][i] == 1) {

*count*++;

**if** (*use*[i] == 0)

*dfs*(i);

}

}

}

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

//Scanner in = new Scanner(System.in);

Scanner in = **new** Scanner(System.*in*);

*n* = in.nextInt();

*g* = **new** **int**[*n*][*n*];

**for** (**int** i=0;i<*n*;i++)

**for** (**int** j=0;j<*n*;j++)

*g*[i][j] = in.nextInt();

*use* = **new** **int**[*n*];

**int** max = 0;

**int** idx = 0;

**for** (**int** i=0;i<*n*;i++)

**if** (*use*[i] == 0) {

*count* = 0;

*mark*++;

*dfs*(i);

**if** (*count* > max) {

max = *count*;

idx = *mark*;

}

}

**for** (**int** i=0;i<*n*;i++)

**if** (*use*[i] == idx)

System.*out*.print(i+" ");

}

}

Перевести арифметическое выражение из инфиксной записи в постфиксную с использованием дерева. Вычислить значение выражения

**import** java.util.Scanner;

**import** java.util.Stack;

**public** **class** Infix2Postfix {

**static** **class** Parser {

**static** **char** *EOT* = '\0';

**int** pos = 0;

**int** len = 0;

String source = "";

**void** resetText(String source) {

pos = 0;

len = source.length();

**this**.source = source;

}

**char** nextChar() {

pos++;

**if** (pos >= len) **return** *EOT*;

**return** source.charAt(pos);

}

**char** curChar() {

**if** (pos >= len) **return** *EOT*;

**return** source.charAt(pos);

}

**void** errorM(String mes) {

System.*err*.println("Parse Error: "+mes);

System.*exit*(1);

}

**void** expected(**char** c) {

**if** ( curChar() != c)

errorM("expected '"+c+"'");

nextChar();

}

}

**static** **class** Node {

**int** type; // store 0 or 1: 0 - operation, 1 - integer;

**char** oper; // operation

**int** value; // integer value

Node l; // left Node

Node r; // right Node

}

**static** Parser *parser*;

**static** Stack<Integer> *stack*;

**static** **int** number() {

**int** res = 0;

**while** (( Character.*isDigit*(*parser*.curChar()) )) {

res \*= 10;

res += Integer.*parseInt*(""+*parser*.curChar()); // cast

*parser*.nextChar();

}

**return** res;

}

**static** Node mult(Node node) {

**if** ( Character.*isDigit*(*parser*.curChar()) ) {

node = **new** Node();

node.type = 1;

node.l = **null**;

node.r = **null**;

node.value = *number*();

**return** node;

}

**else** **if** ( *parser*.curChar() == '(' ) {

*parser*.nextChar();

node = *expr*(node);

*parser*.expected(')');

**return** node;

}

**else** *parser*.errorM("expected number or '('");

**return** **null**;

}

**static** Node term(Node node) {

node = *mult*(node);

**while** (*parser*.curChar() == '\*' || *parser*.curChar() == '/') {

Node t = **new** Node();

t.type = 0;

t.oper = *parser*.curChar();

t.l = node;

*parser*.nextChar();

t.r = *mult*(t.r);

node = t;

}

**return** node;

}

**static** Node expr(Node node) {

node = *term*(node);

**while** (*parser*.curChar() == '+' || *parser*.curChar() == '-') {

Node t = **new** Node();

t.type = 0;

t.oper = *parser*.curChar();

t.l = node;

*parser*.nextChar();

t.r = *term*(t.r);

node = t;

}

**return** node;

}

**static** **void** postfix(Node node) {

**if** ( node == **null** ) **return**;

*postfix*(node.l);

*postfix*(node.r);

**if** (node.type == 0) {

System.*out*.print(node.oper+" ");

**int** a = *stack*.pop();

**int** b = *stack*.pop();

**switch** (node.oper) {

**case** '+': a += b; **break**;

**case** '-': a -= b; **break**;

**case** '\*': a \*= b; **break**;

**case** '/': a /= b; **break**;

}

*stack*.push(a);

}

**else** **if** (node.type == 1) {

System.*out*.print(node.value+" ");

*stack*.push(node.value);

}

}

**public** **static** **void** main(String[] args) {

// init Scanner

Scanner in = **new** Scanner(System.*in*);

// init Parser

*parser* = **new** Parser();

*parser*.resetText(in.nextLine());

// init

/\*

Node root = new Node();

root.l = null;

root.r = null;

\*/

Node root = **null**;

root = *expr*(root);

*stack* = **new** Stack<Integer>();

*postfix*(root);

System.*out*.println("");

System.*out*.println(*stack*.pop());

}

}