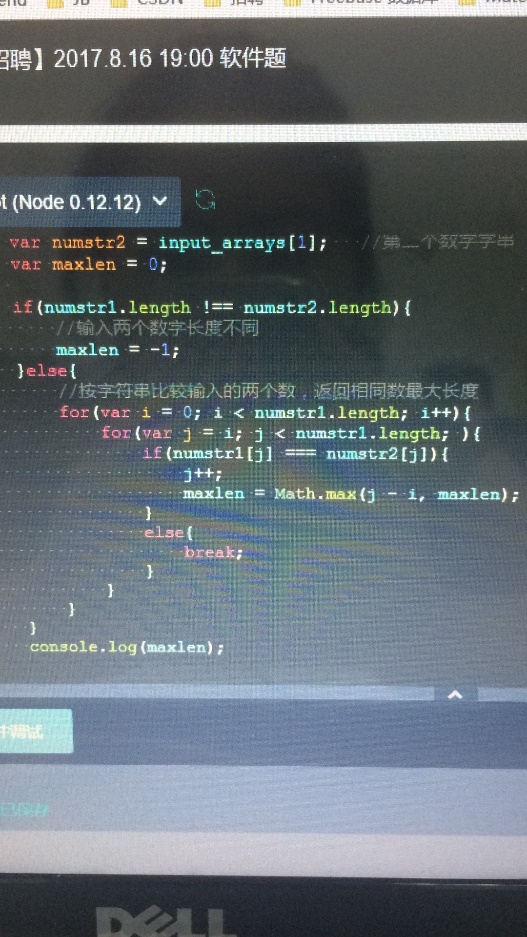
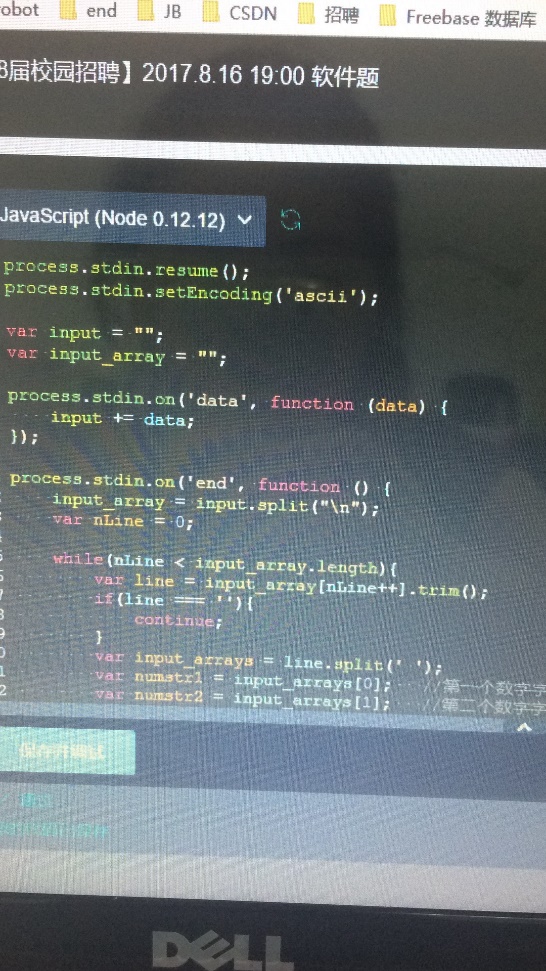
1. 比较两个数列，找出连续相同的最大串长度

1. 判断循环依赖

#include <iostream>

#include <stack>

#define MAXVEX 999999

using namespace std;

typedef struct EdgeNode {

int adjvex;

struct EdgeNode \*next;

}EdgeNode;

typedef struct VertexNode {

int in;

unsigned int data;

EdgeNode \*firstedge;

}VertexNode, AdjList[MAXVEX];

typedef struct {

AdjList adjList;

int num\_vertex;

}GraphAdjList;

GraphAdjList \*graphAdjList = new GraphAdjList();

bool visited[MAXVEX];

int add\_vertex(unsigned int id) {

for (int i = 0; i < graphAdjList->num\_vertex; i++) {

if (graphAdjList->adjList[i].data == id) {

return i;

}

}

int index = graphAdjList->num\_vertex;

graphAdjList->num\_vertex++;

graphAdjList->adjList[index].data = id;

return index;

}

int is\_contained(unsigned int id) {

for (int i = 0; i < graphAdjList->num\_vertex; i++) {

if (graphAdjList->adjList[i].data == id) {

return i;

}

}

return -1;

}

void Add(unsigned int id, unsigned int dependID) {

int id\_index = add\_vertex(id);

int dependID\_index = add\_vertex(dependID);

EdgeNode \*edgeNode = new EdgeNode();

edgeNode->adjvex = dependID\_index;

if (graphAdjList->adjList[id\_index].firstedge == NULL) {

graphAdjList->adjList[id\_index].firstedge = edgeNode;

} else {

edgeNode->next = graphAdjList->adjList[id\_index].firstedge->next;

graphAdjList->adjList[id\_index].firstedge->next = edgeNode;

}

}

void dfs(GraphAdjList \*gl, int i, int index, bool &cycle) {

EdgeNode \*p;

visited[i] = true;

p = gl->adjList[i].firstedge;

while (p) {

if (p->adjvex == index) {

cycle = true;

return;

}

if (!visited[p->adjvex])

dfs(gl, p->adjvex, index, cycle);

p = p->next;

}

}

bool isCycle1(unsigned int id) {

int index = is\_contained(id);

if (index == -1)

return false;

bool cycle = false;

dfs(graphAdjList, index, index, cycle);

return cycle;

}

void output() {

for (int i = 0; i < graphAdjList->num\_vertex; ++i) {

VertexNode vertex = graphAdjList->adjList[i];

EdgeNode \*first = vertex.firstedge;

cout<< "data:" << vertex.data << " ";

while (first) {

cout << first->adjvex << " ";

first = first->next;

}

cout << endl;

}

}

int main() {

int count = 5;

unsigned int num1, num2;

for (int i = 0; i < count; ++i) {

cin >> num1 >> num2;

Add(num1, num2);

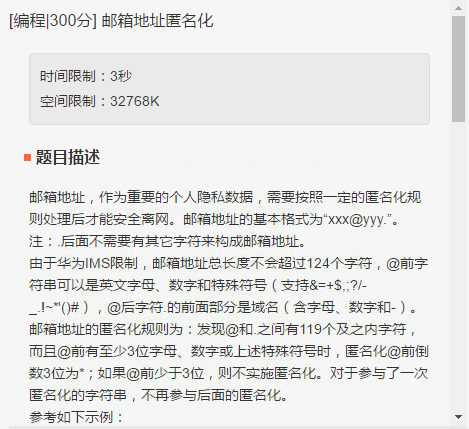
}

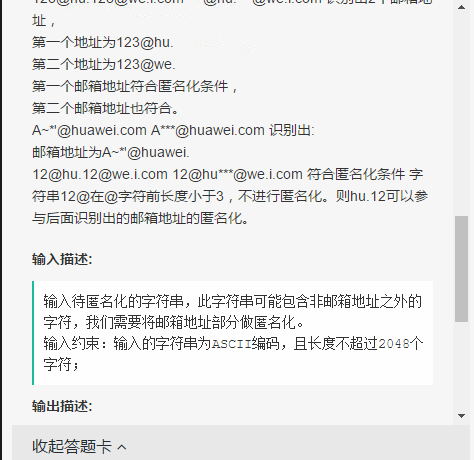
cout << isCycle1(1) <<endl;

output();

}

1. 正则表达式判断





import java.util.\*;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in

);

String string = sc.nextLine();

Pattern pattern = Pattern.compile("[a-zA-Z0-9&=\\+\\$,;\\?/\\-\_\\.!~\\\*'\\(\\)#]{3,}@[a-zA-Z0-9\\-]+\\.");

Matcher matcher = pattern.matcher(string);

StringBuffer anwser = new StringBuffer();

List<Integer> beginning = new ArrayList<>();

List<Integer> ending = new ArrayList<>();

List<String> mailname = new ArrayList<>();

while (matcher.find()) {

int start = matcher.start();

int end = matcher.end();

String mail = matcher.group();

String[] s = mail.split("@");

String yuming = s[1].substring(0, s[1].length() - 1);

if (end - start <= 124 && yuming.length() <= 119 && !mailname.contains(yuming)) {

mailname.add(yuming);

beginning.add(start);

ending.add(end);

}

}

int start = 0;

for (int i = 0; i < beginning.size(); i++) {

String sub = string.substring(start, beginning.get(i));

anwser.append(sub);

String[] temp = string.substring(beginning.get(i), ending.get(i)).split("@");

String fistf = temp[0].substring(0, temp[0].length() - 3);

anwser.append(fistf + "\*\*\*@" + temp[1]);

if (i != beginning.size() - 1) {

String last = string.substring(ending.get(i), beginning.get(i + 1));

anwser.append(last);

start = beginning.get(i + 1);

} else {

String last = string.substring(ending.get(i), string.length());

anwser.append(last);

}

}

if (beginning.size() == 0)

System.out.println(string);

else

System.out.println(anwser.toString());

}

}