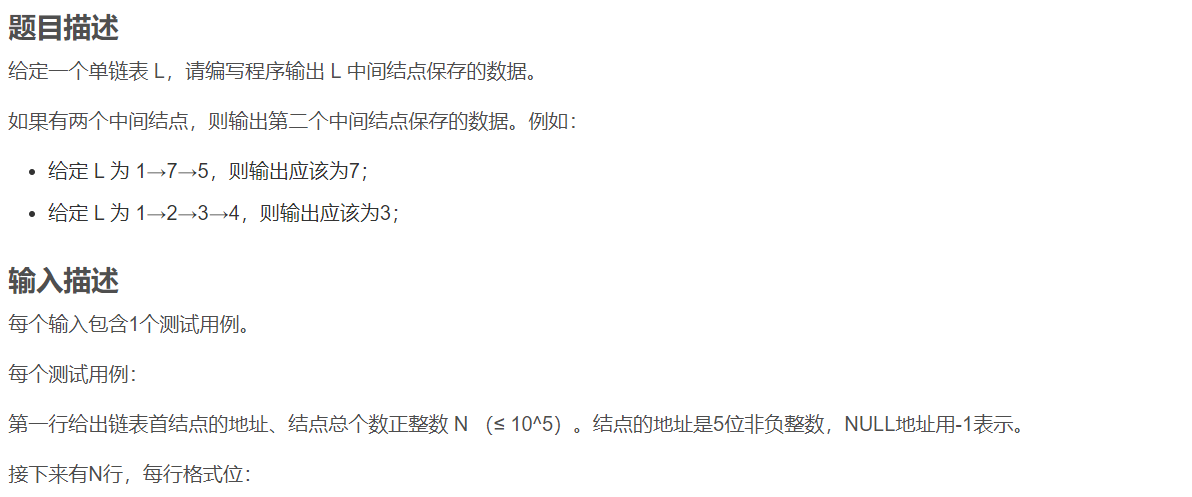
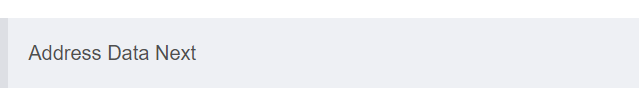
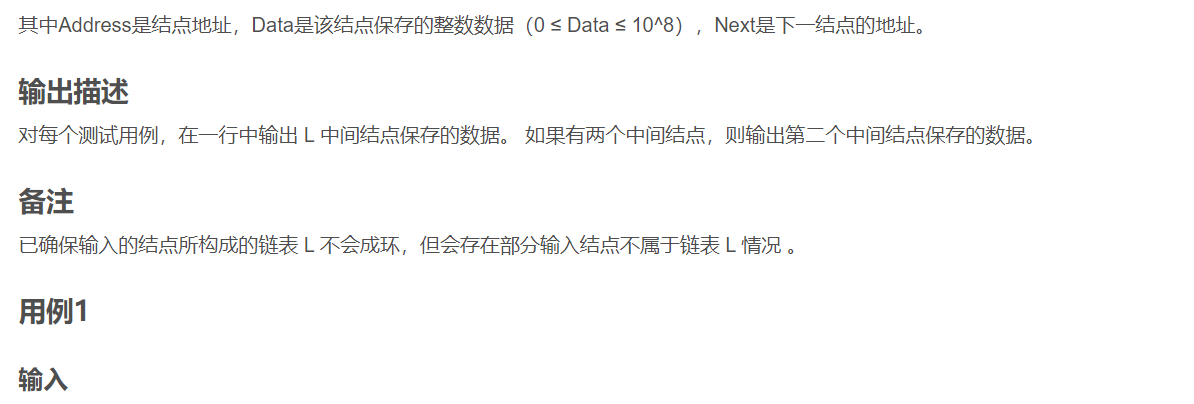
# **E卷-寻找链表的中间结点[100分]（ Java | Python3 | C++ | C语言 | JsNode | Go ）**







00010 4

00000 3 -1

00010 5 12309

11451 6 00000

12309 7 11451



6



10000 3

76892 7 12309

12309 5 -1

10000 1 76892



7



import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// 使用一个哈希表来存储链表的结点信息

HashMap<String, String[]> mp = new HashMap<>();

// 读取链表的首结点地址

String head = sc.next();

// 读取结点的总个数

int n = sc.nextInt();

// 读取每个结点的信息

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

String add = sc.next(); // 结点地址

String val = sc.next(); // 结点保存的数据

String next = sc.next(); // 下一结点的地址

mp.put(add, new String[] { val, next });

}

// 计算链表的长度

int length = 0;

String tmp = head;

while (tmp.compareTo("-1") != 0) { // "-1"表示链表的结尾

length += 1;

tmp = mp.get(tmp)[1]; // 获取下一个结点的地址

}

// 计算中间结点的索引

int cnt = length / 2;

// 找到中间的结点

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

head = mp.get(head)[1];

}

// 输出中间结点的数据

System.out.println(mp.get(head)[0]);

}

}



# 使用字典来存储链表的结点信息

nodes = {}

# 读取链表的首结点地址和结点的总个数

head, n = input().split()

n = int(n)

# 读取每个结点的信息

for \_ in range(n):

addr, data, next = input().split()

nodes[addr] = {"data": data, "next": next}

# 计算链表的长度

length = 0

tmp = head

while tmp != "-1":

length += 1

tmp = nodes[tmp]["next"]

# 计算中间结点的索引

cnt = length // 2

# 找到中间的结点

for \_ in range(cnt):

head = nodes[head]["next"]

# 输出中间结点的数据

print(nodes[head]["data"])



#include <iostream>

#include <unordered\_map>

#include <string>

using namespace std;

// 定义一个结构体来存储每个结点的信息

struct Node {

string data; // 结点保存的数据

string next; // 下一结点的地址

};

int main() {

int n;

string head;

cin >> head >> n;

unordered\_map<string, Node> mp;

// 读取每个结点的信息

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

string addr, next;

string data;

cin >> addr >> data >> next;

mp[addr] = { data, next };

}

// 计算链表的长度

int length = 0;

string tmp = head;

while (tmp != "-1") {

length += 1;

tmp = mp[tmp].next;

}

// 计算中间结点的索引

int cnt = length / 2;

// 找到中间的结点

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

head = mp[head].next;

}

// 输出中间结点的数据

cout << mp[head].data << endl;

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// 定义一个结构体来存储每个结点的信息

typedef struct Node {

char data[11]; // 结点保存的数据

char next[6]; // 下一结点的地址

} Node;

// 哈希表的最大大小设为100000（根据题目要求）

#define MAXN 100000

int main() {

int n;

char head[6];

// 读取链表的首结点地址和结点总个数

scanf("%s %d", head, &n);

// 创建一个哈希表来存储链表的结点信息

Node hashMap[MAXN];

char addresses[MAXN][6];

// 读取每个结点的信息

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

char addr[6], next[6];

int data;

scanf("%s %d %s", addr, &data, next);

strcpy(addresses[i], addr);

sprintf(hashMap[i].data, "%d", data);

strcpy(hashMap[i].next, next);

}

// 计算链表的长度

int length = 0;

int index = -1;

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

if (strcmp(addresses[i], head) == 0) {

index = i;

break;

}

}

// 遍历链表以计算长度

while (index != -1 && strcmp(hashMap[index].next, "-1") != 0) {

length += 1;

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

if (strcmp(addresses[i], hashMap[index].next) == 0) {

index = i;

break;

}

}

}

length += 1; // 包含最后一个结点

// 计算中间结点的索引

int cnt = length / 2;

// 找到中间结点

index = -1;

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

if (strcmp(addresses[i], head) == 0) {

index = i;

break;

}

}

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

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

if (strcmp(addresses[j], hashMap[index].next) == 0) {

index = j;

break;

}

}

}

// 输出中间结点的数据

printf("%s\n", hashMap[index].data);

return 0;

}



const readline = require('readline');

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

let input = [];

rl.on('line', (line) => {

input.push(line);

}).on('close', () => {

let mp = {};

// 读取链表的首结点地址和结点总个数

let [head, n] = input[0].split(' ');

n = parseInt(n);

// 读取每个结点的信息

for (let i = 1; i <= n; i++) {

let [addr, data, next] = input[i].split(' ');

mp[addr] = { data: data, next: next };

}

// 计算链表的长度

let length = 0;

let tmp = head;

while (tmp !== "-1") {

length += 1;

tmp = mp[tmp].next;

}

// 计算中间结点的索引

let cnt = Math.floor(length / 2);

// 找到中间的结点

for (let i = 0; i < cnt; i++) {

head = mp[head].next;

}

// 输出中间结点的数据

console.log(mp[head].data);

});



package main

import (

"bufio"

"fmt"

"os"

"strconv"

"strings"

)

// 定义一个结构体来存储每个结点的信息

type Node struct {

data string

next string

}

func main() {

reader := bufio.NewReader(os.Stdin)

// 读取链表首结点的地址和结点总个数

firstLine, \_ := reader.ReadString('\n')

firstLine = strings.TrimSpace(firstLine)

parts := strings.Split(firstLine, " ")

head := parts[0]

n, \_ := strconv.Atoi(parts[1])

// 创建一个哈希表储存链表结点信息

mp := make(map[string]Node)

// 读取每个结点的信息

for i := 0; i < n; i++ {

line, \_ := reader.ReadString('\n')

line = strings.TrimSpace(line)

parts := strings.Split(line, " ")

addr := parts[0]

data := parts[1]

next := parts[2]

mp[addr] = Node{data: data, next: next}

}

// 计算链表的长度

length := 0

tmp := head

for tmp != "-1" {

length++

tmp = mp[tmp].next

}

// 计算中间结点的索引

cnt := length / 2

// 找到中间的结点

for i := 0; i < cnt; i++ {

head = mp[head].next

}

// 输出中间结点的数据

fmt.Println(mp[head].data)

}